University of California
Los Angeles

Avoidance Routing

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

by

Erik Alan Kline

2012

Abstract of the Dissertation

Avoidance Routing

by

Erik Alan Kline

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2012

Professor Mario Gerla, Co-chair

Professor Peter Reiher, Co-chair

In this dissertation, we describe avoidance routing. Avoidance routing is a mechanism that provides users control over how their data transits the Internet. Currently, a user may use cryptography to provide confidentially for their data, but has no control over who can observe or access their data. Avoidance routing provides this control by having users specify properties of routers they do not trust, such as geopolitical location. A path to the destination that does not transit routers with the given properties is found, on-demand, with reasonable assurances that the path found does not contain untrusted routers.

Avoidance routing accomplishes this by augmenting BGP with a security vector. This vector contains the properties of each router along the given AS path. Given this vector and a users criteria, avoidance routing conducts a depth-first search to find a valid path the destination. We leverage the information contained within BGP to try and find this path quickly and optimally. Our results show that avoidance routing tends to find short paths, on average 7 hops longer than the standard path in the worst case. Further, less than 1% of nodes on the Internet are visited, and only 5% in the worst case. The storage space required to store the augmented advertisements or conduct the search is both reasonable and small. Finally, their is no performance impact in terms of packet forwarding by using avoidance routing.

We will discuss our use of a consensus mechanism to provide protection of routing information. This mechanism treats each neighbor of a node as a consensus speaker or "voter". These speakers will each make claims about their neighbor, and the consensus approach will attempt to determine the validity of these claims as well as the quality of each speaker. When routing, only nodes with high agreement on their claims will be used, as well as only speakers with high quality. Our results show that our consensus mechanism, while simple, also provides reasonable security guarantees.

The dissertation of Erik Alan Kline is approved.

Ying Nian Wu

Douglas Stott Parker

Leonard Kleinrock

Peter Reiher, Committee Co-chair

Mario Gerla, Committee Co-chair

University of California, Los Angeles

2012

I dedicate this thesis to my friends and family who have supported me throughout the process. I would especially like to thank my parents for nurturing my drive for knowledge without attempting to control my future. I'd also like to thank all of my grandparents for inspiring me in different ways. My grandmother and grandfather, for pushing education and knowledge above all else. My grandma and grandpa, for making sure I don't treat life too seriously. My sister has also been extremely helpful by showing me that there will always be peope smarter or better than me at something.

My advisor, Peter, has also been extremely helpful at keeping me focused. He was able to help guide me through the desert as I wandered. When I finally found an idea to pursue, he was able to give me excellent advice to help me reach this conclusion. Further, my brothers in arms in graduate school were extremely helpful. It is always nice to know that we are all in this together. I'd like to thank Janice, for her countless hours reading and editting both my thesis and papers.

Finally, I'd like to thank my girlfriend Stephanie. She has stood by me while I worked tirelessly to finish my thesis. Her encouragement and drive helped me finally finish this work. Further, she put up with the late nights and long weekends while I worked without complaining and with constant enthusiasm and encouragement. For all of these people, I dedicate this thesis.

Table of Contents

1 Introduction **1**

1.1 What is Avoidance Routing? 5

2 Control over Routing **8**

2.1 Existing Solutions 8

2.2 Internet Topology 12

2.2.1 Annotating the Topology 16

2.2.2 Disclosure Limitations 17

2.3 Security, Scalability and Speed 17

2.4 Trust 19

3 Annotating the Topology **21**

3.1 Obtaining a Topology 22

3.2 Security Properties 26

3.2.1 Geopolitical Location 27

3.2.2 Corporate Ownership 28

3.2.3 Firmware Version and Hardware Manufacturer 28

3.2.4 Autonomous System 28

3.2.5 Reputation 29

3.2.6 Path Attribution 29

3.2.7 Consensus and Agreement 29

3.3 Property Changes 30

3.4 Determining Security Properties of Routers 31
3.4.1 Property Classes 32
3.4.2 Incentivizing Disclosure 33
3.5 Dissemination of Security Information 34
3.5.1 Partial Disclosure 37
3.5.2 Advertisement Overhead 37

4 Routing Using Security Properties 41
4.1 Avoidance Requests 41
4.2 Depth-First Search 45
4.2.1 Caching 51
4.2.2 Path Invalidation 54
4.3 Policy 59
4.4 Heuristics 62
4.4.1 Neighbor Only 63
4.4.2 Shortest Path First 64
4.4.3 Furthest Collision 65
4.4.4 Nearest Collision 65
4.4.5 Least Collisions 66
4.4.6 Prioritized Heuristics 66
4.5 Example of Avoidance Routing 67
4.6 Security 72
4.6.1 Using the Search for Denial-of-Service 73
4.6.2 Rate Limiting 74

4.6.3 Avoidance Message Spoofing 80
4.7 Usability 82
4.8 Partial Deployment 84
4.9 Intra-AS Avoidance Routing 84
4.10 Avoidance vs. Allowance 86

5 Evaluation of the Avoidance Routing Search 87
5.1 Software Implementation 87
5.2 Avoidance Routing Simulator 89
5.3 Available Paths 93
5.4 Path Length 98
5.5 Extraneous Nodes Visited 117
5.6 Discussion 127
5.7 Search Overhead 137
5.8 Evaluation of Forwarding 139
5.8.1 Non-avoidance Packets 139
5.8.2 Avoidance Packets 141

6 Trusting Security Information 146
6.1 Possible Protection Mechanisms 147
6.1.1 Authentication 147
6.1.2 Validation 149
6.1.3 Reputation 150
6.1.4 Consensus 151
6.2 Using Consensus for Trust 153

6.2.1 Agreement . . . 154
6.2.2 Judgment . . . 156
6.2.3 Normalized Judgment . . . 158
6.2.4 Volatility . . . 159
6.2.5 Confidence . . . 160
6.2.6 Consensus in Use . . . 161
6.2.7 An Example of Using Consensus . . . 162
6.3 Partial Disclosure . . . 167
6.4 Evaluation of Consensus . . . 168
6.5 Limitations . . . 175

7 Future Work . . . 178
7.1 Ensuring Correct Routing . . . 179
7.1.1 Observation . . . 179
7.1.2 Construction . . . 180
7.2 Search Optimizations . . . 181
7.2.1 Partial Map Construction . . . 183
7.2.2 Parallel Searches . . . 185
7.2.3 Other Search Algorithms . . . 186
7.3 Geo-mapping . . . 187
7.4 Receiver-Oriented Avoidance Routing . . . 187

8 Related Work . . . 189
8.1 Alternate Approaches . . . 189
8.2 Trust-Based Routing Protocols . . . 193

8.3 Geographic-Based Routing 194
8.4 Routing via Search 195
8.5 Policy-Based Routing 196
8.6 Multi-Protocol Label Switching 198
8.7 Consensus 199

9 Conclusion **201**

References **206**

List of Figures

1.1 A simple topology overlayed on a map of Europe 6

2.1 A simple six-node topology 13

2.2 The simple topology as constructed from advertisements 14

2.3 A topology showing the Internet hierarchy 15

2.4 The original European topology 19

3.1 A simple network topology 35

3.2 A simple topology showing bidirectional claims 39

4.1 The header form layout 42

4.2 The packet form layout 43

4.3 The IPv6 header form layout 45

4.4 Our simple six-node topology 46

4.5 Node A's view of the topology 47

4.6 The search table entry 50

4.7 The header form 53

4.8 The no-valley policy 60

4.9 Favor customers 61

4.10 An example topology for avoidance routing 67

4.11 The routes to node D that nodes 1 and 2 are aware of 69

4.12 The routes to node D that node 8 is aware of 70

4.13 Distribution of rate limits for all ASes propagated top-down 75

4.14 Distribution of rate limits for all ASes propagated bottom up 76

4.15 Maximum and average available rate with full-rate attack 78
4.16 Maximum and average available rate with the spread Attack 79
4.17 Effect on the targeted AS during the different attacks 80
4.18 Interfaces for legacy and modern applications 84

5.1 A simple topology where "favor customers" breaks 92
5.2 Connectivity of remaining graph after uniform removal 94
5.3 Connectivity of remaining graph after half-Gaussian removal 94
5.4 Connectivity of remaining graph after geopolitical removal from largest to smallest 97
5.5 Connectivity of remaining graph after geopolitical removal from smallest to largest 98
5.6 Increase in path length with no policy using uniform properties 100
5.7 Increase in path length with no policy using half-Gaussian properties . . . 100
5.8 Increase in path length with no policy using geopolitical properties from largest to smallest 102
5.9 Increase in path length with no policy using geopolitical properties from smallest to largest 102
5.10 Increase in path length with policy using uniform properties 103
5.11 Increase in path length with policy using half-Gaussian properties 104
5.12 Increase in path length with policy using geopolitical properties from largest to smallest 105
5.13 Increase in path length with policy using geopolitical properties from smallest to largest 105
5.14 Increase in path length for uniform properties when favoring customers . . 106

5.15 Increase in path length for half-Gaussian properties when favoring customers 107

5.16 Increase in path length with policy using geopolitical properties from largest to smallest while favoring customers 107

5.17 Increase in path length with policy using geopolitical properties from smallest to largest while favoring customers 108

5.18 Increase in path length for a uniform property distribution when favoring customer 109

5.19 Increase in path length and error with no policy using uniform properties . 110

5.20 Increase in path length and error with no policy using geopolitical properties 110

5.21 Increase in path length and error with no-valley policy using uniform properties 111

5.22 Increase in path length and error with no-valley policy using geopolitical properties 111

5.23 Increase in path length and error with favor customer policy using uniform properties 112

5.24 Increase in path length and error with favor customer policy using geopolitical properties 112

5.25 CDF of path length increase using uniform property distribution and "no-valley" policy 114

5.26 CDF of path length increase using uniform property distribution and favoring customers 114

5.27 CDF of path length increase using geopolitical (largest) property distribution and "no-valley" policy 116

5.28 CDF of path length increase using geopolitical (largest) property distribution and favoring customers 116

5.29 Extraneous nodes visited for the uniform property distribution and the "no valley" policy 119
5.30 Extraneous nodes visited for the half-Gaussian property distribution and the "no valley" policy 119
5.31 Extraneous nodes visited for the geopolitical property distribution and the "no valley" policy 120
5.32 Extraneous nodes visited for the uniform property distribution and favoring customers 121
5.33 Extraneous nodes visited for the half-Gaussian property distribution and favoring customers 121
5.34 Extraneous nodes visited for the geopolitical property distribution and favoring customers 122
5.35 Extraneous nodes visited with policy for the uniform property distribution 123
5.36 CDF of extraneous nodes visited using uniform property distribution and "no-valley" policy 125
5.37 CDF of extraneous nodes visited using uniform property distribution and favoring customers 125
5.38 CDF of extraneous nodes visited using geopolitical property distribution and "no-valley" policy 126
5.39 CDF of extraneous nodes visited using geopolitical property distribution and favoring customers 126
5.40 Percentage of avoidance routes discovered by a valid advertisement within the first two hops, with the uniform property distribution 128
5.41 Percentage of avoidance routes discovered by a valid advertisement within the first two hops, with the geopolitical property distribution 129

5.42 Increase in Path Length when chaining heuristics 130
5.43 Topology when avoidance routing is better than BGP 132
5.44 Percentage of routes that are the same as the standard route with the uniform property distribution 134
5.45 Percentage of routes that are the same as the standard route with the geopolitical property distribution 134
5.46 Connection duration for non-avoidance routed packets 140
5.47 Time to first byte for non-avoidance routed packets 140
5.48 Connection duration for avoidance routed packets 142
5.49 Time to first byte for avoidance routed packets 142

6.1 A simple topology where colluders may force a incorrect decision 152
6.2 Simple topology where nodes are stating their beliefs about node X 155
6.3 In this topology, each letter node is making a claim about a numbered node, where its belief is shown in the annotated link 157
6.4 A simple topology where S wants to communicate with D 163
6.5 The agreement on all valid nodes in the topology 164
6.6 The judgment of all nodes in the topology 165
6.7 The agreement on all valid nodes in the topology with judgment 166
6.8 Strength of belief as number of dissenters increase 168
6.9 Judgment as the number of dissenters increase 170
6.10 Agreement as the number of dissenters increase including normalized judgment 171
6.11 Average strength of belief of the neighbors of each country 172
6.12 Average strength of belief using normalized judgment 173

6.13 Change in confidence as number of dissenters increases 174

6.14 Average confidence of the neighbors of each country 175

7.1 Simple topology illustrating when avoidance routing may perform poorly . 182

7.2 Topology illustrating information gleamed from multiple advertisements . . 184

List of Tables

3.1 Growth of advertisement with or without agreement 40

4.1 Increase of maximum and average rate limit as minimum grows 77

4.2 Standard functions and avoidance counterparts 83

5.1 Maximum observed path for different property distributions (Percent of graph removed) 113

5.2 Maximum extraneous nodes visited for different property distributions (Percent of graph removed) 124

Vita

2003–2005	Computer Science Intern, Food Processing Technology Division Georgia Tech Research Institute, Atlanta, Georgia.
2005	B.S. (Computer Science), Georgia Institute of Technology, Atlanta, Georgia.
2005–2006	Research Assistant, Laboratory for Advanced Systems Research, Computer Science Department, UCLA.
2006–2007	Teaching Assistant, Computer Science Department, UCLA. Taught sections of CS 31 and CS 32 (Introduction to Computer Science).
2007–2009	Research Assistant, Laboratory for Advanced Systems Research, Computer Science Department, UCLA.
2009	M.S. (Computer Science), UCLA, Los Angeles, California.
2010–2012	Research Assistant, Computer Networks Division, Information Sciences Institute, USC.
2011	Teaching Assistant, Computer Science Department, UCLA. Taught CS 136, Computer Security
2012–present	Computer Scientist, Computer Networks Division, Information Sciences Institute, USC.

Publications

Erik A. Kline, and Peter Reiher. Securing Data through Avoidance Routing. In *NSPW*, September 2009

Erik A. Kline, Matthew Beaumont-Gay, Jelena Mirkovic, and Peter Reiher. RAD: Reflector Attack Defense using Message Authentication Codes. In *ACSAC*, December 2009

Erik A. Kline, Alexander Afanasyev and Peter Reiher. Shield: DoS Defense using Traffic Deflection In *FIST*, October 2011

Charles Fleming, Peter Peterson, Erik A. Kline, and Peter Reiher. Data Tethers. In *ICC*, May 2012

CHAPTER 1

Introduction

Throughout history, organizations, governments and individuals have gone to extensive lengths to protect their important assets. Much of this effort went into protecting physical assests, such as money and property. However, equally or even more important is the protection of one's information. In ancient times, knowing the location of resources might allow one civilization to succeed where another failed. There are numerous examples where critical military information might have allowed a player to gain the upper hand over its adversary. Had the United States been aware of the impending Japanese attack on Pearl Harbor, the outcome would assuredly have been different. If the traitor Ephialtes had not told Xerxes of the pass above Thermopylae, would the battle have gone differently? The "what if" game can be played ad nauseum, but it is clear from these examples that information, or the lack thereof, plays a critical role.

In modern times military information is only one type of critical information. Corporations go to extensive lengths to protect both their trade and financial secrets. The Coca-Cola company, for example, has famously gone to ridiculous efforts to keep the Coke formula a secret. Further, inside information is so valuable that it has the power to cause massive stock market fluctuations if made public. Protecting corporate information may be the difference between success and bankruptcy.

Individuals also need to protect their information. The growth of the Internet has allowed an increase in identity theft, making people more wary about the disclosure of personal information. Banking and credit information may be used to steal money or make

illegitimate transactions. Some information can be used to stalk a person, and possibly lead to worse outcomes [Bil12]. Intellectual and innovative ideas are also property that one might want to protect in order to profit from them. Failure to do so may result in loss of wealth, fame or business opportunities.

The modern age has produced many technological wonders which offer us many benefits. Unfortunately, these technologies also create new ways for people to steal and make use of our information. Many governments, such as the United States and Sweden [PAA07] [BBC08], have laws allowing wire taps. There have been reports of Chinese efforts to conduct computer-based espionage [RD09] [Gor11]. Recent reports of corporate espionage [Lew08] [Jus08] continue to reinforce the need for information protection. Finally, there is a multitude of malicious software that attempts to steal people's credentials [Woo12] in order to make a profit [MCA09].

Clearly, protecting information is critical to many different entites at many different levels. The obvious question is, "How do we protect our information?". There are two core mechanisms that can be used. The first method is to make the information unusable even if obtained by an adversary. The second mechanism is to simply prevent an untrusted or adversarial party (hereafter referred to as *an adversary*) from obtaining the information. In general, this is referred to as "access control."

The first technique has been in use since ancient times and generally falls into the realm of cryptography. Herodotus writes of two examples of steganography in use since at least 440 BC in his book, The Histories of Herodotus [HerBC]. Other forms of cryptography, such as the substitution ciphers, have been in use for over a thousand years [Sue21] [Sch96]. In modern times, cryptography is a very common way to protect information.

Other mechanisms besides cryptography can also make information unusable. False or misinformation is a viable strategy that has often been used. In the Second World War, the Allies used several deception strategies to confuse the Germans about when and where D-Day would occur [Hes00]. It is assumed that an adversary cannot tell the difference

between real and false information; thus an adverary must treat all information acquired as suspect.

The second technique seems to be the more obvious approach – prevent an enemy from gaining access to your sensitive information. Keeping sensitive information in the hands of a few is one method of protecting it. Only these few individuals need to be trusted. This is essentially access control, i.e. restricting those who have access to the information. Trusted couriers have been used to protect the transit of sensitive information, preventing it from traveling through untrusted regions or falling into the hands of an adversary. In modern times, access control is the primary method used to prevent an adversary from obtaining data.

The modern era has introduced digitial information, but the same general techniques are used to protect our information. Access control in operating systems restricts users and processes from reading or writing data. Cryptography is also used to make the data unusable except to those who have access to it.

This is only true of data on a machine we control. Once the data has been transmitted onto the Internet, it is out of our control. Who can obtain and read our data as it transits the Internet is something many users would like to control. Cryptography is still a valid method to protect our data, but it provides no control over who can obtain the data. If the cryptography can be broken, the adversary has won.

History is rife with cases where one party was able to successfully break the cryptography of another [Gut02][Kah67][Sin00]. Further, one does not necessarily need to break the cryptography to gain useful intelligence. Traffic patterns may allow inference on traffic type and content [Sch00][WMS11]. Even supposedly anonymous communication technologies, which heavily use cryptography, have been the vulnerable to correlation attacks which lead to de-anonymization [WCJ05][ZFG04].

It is clear that cryptography is an insufficient solution. True security can only be obtained if we can use both mechanisms, cryptography *and* access control. Reducing your

adversary's access to your encrypted data can significantly increase its safety. Historically, adversaries have made effective use of intercepted encrypted data. British intelligence's ability to intercept and decrypt German communications in both World War I and World War II was a great boon to the allied war efforts. The intercepted and decrypted Zimmerman Telegram, for example, enabled the British to influence the United States to enter World War I [Tuc66].

A mechanism to control who has access to our data as it transits the Internet is required. Given the current architecture of the Internet, we require a mechanism that controls which routers our data transits. With this mechanism, we can control which entities will have access to our data.

The security value of controlling our Internet routing paths has been recognized [Sch08], but there are currently no feasible methods to facilitate this control. At best, we can determine the route that will be taken, although this is not guaranteed. While source routing [Pos81] would allow one to force a specific route, it is not clear how one would go about determining this route. Further, relatively few Internet routers support source routing [GT00].

The only current approach is to have service providers and autonomous systems manually set up specific routes via Border Gateway Protocol (BGP) policies and intelligent Multiprotocol Label Switching (MPLS) internally. This is generally done for traffic engineering and quality-of-service purposes, but not for security reasons. The inability to control who can view our information also makes it difficult to disseminate false information, since the goal is for the enemy to obtain it. A practical method for avoiding specific regions of the Internet is needed for those who require better security for sensitive data.

In this dissertation we will discuss *avoidance routing*, a system that provides users with finer control over how their information is routed. Users will be able to specify properties of routers that they deem untrustworthy, with reasonable assurances that their wishes will be met. These properties include, but are not limited to, geopolitical location, corporate

ownership, router type or manufacturer, and specific autonomous systems (ASes).

The Internet is currently made up of multiple ASes, networks where internally everything is under one administrative domain. These systems peer with each other, resulting in a decenteralized control of the Internet. Therefore, ensuring that each AS properly uses avoidance routing is a difficult problem that must be solved for the system to work. This challenge and its solution will be discussed in more detail in Chapter 6.

To make avoidance routing usable in the current Internet, we designed it to be compatible with existing routing protocols, and we will discuss how this was accomplished. Further, we will show that this method will incur low additional routing costs and achieve reasonable scaling. Low overhead will be accomplished while still allowing many different parties to use avoidance routing, specifying different sets of properties that they wish to avoid when sending data. Some challenges that avoidance routing must overcome are discussed in Chapter 2. The avoidance routing design will be specified in detail in Chapter 3. Further details of specific aspects, as well as their evaluation, can be found in Chapters 4, 5 and 6.

1.1 What is Avoidance Routing?

The stated goal of avoidance routing is to *provide a mechanism for users to specify properties of routers they do not trust. Given this criteria, a route that avoids untrusted routers will be found. The user's traffic will be sent along the appropriate route, giving the user reasonable assurances that his data will not transit untrusted nodes.* This goal, in our opinion, is both lofty and reasonable. One might ask, "What does it mean?"

Consider the simple topology in Figure 1.1, which has been conveniently overlayed on a map of Europe [Fre12]. In this example, sender S wants to reach D by avoiding the geopolitical security property of Germany. Currently, S has no control over routing, so when their data reaches node A in France, it may be given to node B in Germany. The

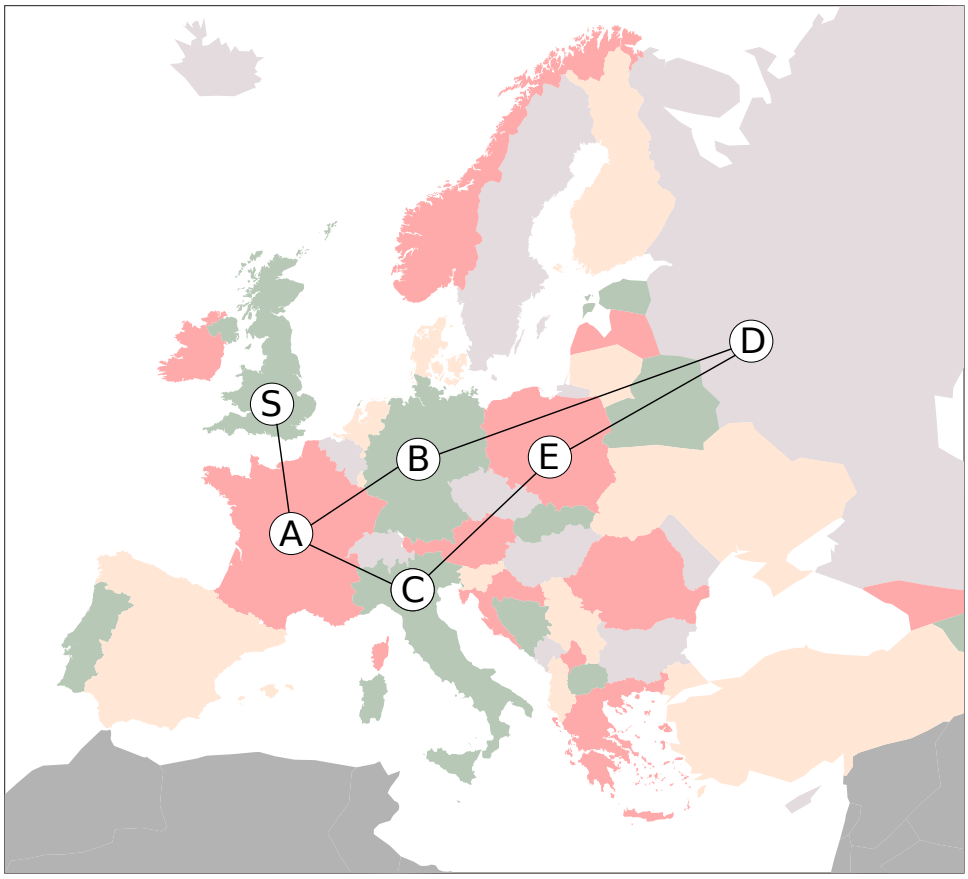

Figure 1.1: A simple topology overlayed on a map of Europe

$B \Rightarrow D$ route is shorter than the $C \Rightarrow E \Rightarrow D$ router and might be preferred because of this. Avoidance routing will provide a mechanism so that his data does not pass through router B.

A simple solution is to build a routing protocol that selectively avoids Germany; however, this is not practical. Multiple different users may have multiple different security criteria. For example, the same sender S may want to send some data that avoids Italy. We would have to build another protocol that selectively avoided Italy. Instead, avoidance routing is designed to allow users to specify any avoidance criteria and find routes that match these criteria.

Further, a route found for one user will not restrict another. While S may wish to use

the Germany avoiding route, other users may not care. These other users' primary concern may be performance, where the shortest route, not the avoidance route, is the one they would like. Thus, avoidance routing needs to work only for users who wish to use it, and leave standard routing unaffected.

Formally, the avoidance routing mechanism that we have designed and implemented allows multiple users to request multiple avoidance routing paths on-demand. These requests do not pigeon hole the mechanism into specific paths, but allow all paths to used simultaneously. Most importantly, the mechanism does not negatively affect standard (non-avoidance routed) traffic in any way. The mechanism that we will discuss does not make any modifications to standard routing pathways, either software or hardware, and thus should not have a negative effect on traffic.

CHAPTER 2

Control over Routing

In Chapter 1, we show that the Internet lacks a usable method for providing end-hosts with control over routing. While a couple of options do exist, such as source routing and overlay networks, these mechanisms have serious limitations. In fact, there are a series of significant problems that need to be overcome for avoidance routing to work. In this chapter we will describe the challenges that avoidance routing must overcome to be a practical and usable technology. To illustrate these challenges, we will first discuss the limitations of existing technologies, showing that many of these limitations are the same that avoidance routing must overcome.

2.1 Existing Solutions

What appears to be the simplest way to accomplish avoidance routing without modifications to the Internet is source routing. Source routing, as currently deployed, is specified as a option in the Internet Protocol version 4 specification. As described in RFC 791 [Pos81], source routing "provides a means for the source of an Internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination." In essence, it accomplishes the goal of avoidance routing by allowing the sender to explicitly specify the route a packet should take.

Source routing works by adding a list of IP address to the IP header as an option. In a version of source routing called loose source routing, each address specifies the next destination router. When a router receives the packet, it routes it towards the current destination

router. How the packet reaches the next destination router is up to the underlying routing protocol. When the current destination router receives the packet, it moves on to the next destination on the list. One can view loose source routing as a bus with a list of stops on its way to the final destination. The bus will travel to each stop in order, but how it gets from stop to stop is not necessarily specified.

Another version of source routing, as specified in RFC 791, is strict source routing. Strict source routing does not provide the same flexibility. In strict source routing, each hop must be specified in the list of addresses. When a router receives a strict source routing packet, it determines which of its neighbors is specified next in the packet and sends it to that neighbor. If no neighbor is specified, the packet is discarded. In this way, the sender must specify the entire route.

It should be noted that the above is source routing as described in RFC 791. Internet Protocol version 6, specified in RFC 2460 [DH98], also includes a source routing option. Unlike IPv4, it does not include a strict source routing option. There may be other ways to design and implement source routing; however they will all tend to suffer from several limitations.

The most fundamental limitation of source routing is determining the Internet topology. In order for a end host to construct a source routed packet, it must first know the route it wants the packet to take. This construction requires the end host to have some partial knowledge of the Internet topology. In the case of avoidance routing, the annotated topology is critical in order to find a route that meets the end host's constraints. However, it has been shown that accurately determining the topology of the Internet is extremely difficult [OPW10].

Another significant problem with source routing is the lack of deployment. In 2000, Govindan [GT00] found that only 8% of routers were source-routing capable. There is little reason to believe this number has gone up, since there are many reasons to not deploy source routing. The first is that source routing is slow. Source routing requires a lot of

processing per packet. In order to make avoidance routing usable, we will show that we can overcome this limitation.

The other critical reason to not deploy source routing is that it can be used for interesting new attacks [Hoe09][Cor08]. These attacks work by exploiting loose source routing. Since the sender specifies the list of stops, the attacker can spoof an address while adding his own address to the list of stops. The receiver believes that the packet came from the spoofed address and sends a reply; however the receiver will send the reply along the same source-routed path. Since the attacker is on that path, the attacker will receive the reply and be able to use it. Source routing in IPv6 suffers from the same problem [Bei07], which is fundamentally an IP spoofing problem.

This allows an attacker to have a new set of IP spoofing attacks. Current IP spoofing attacks allow an attacker to hide his identity by lying about his IP address. However, doing so does not allow the attacker to receive any of the responses to his messages because the responses will go to the spoofed sender. This restricts the type of attacks available to an attacker, generally to only flooding attacks. However, using the technique described above, an attacker can receive the responses sent by the victim, and respond to them. This allows an attacker to open full TCP sessions since he can respond to TCP control messages. Further, an attacker can even request and receive data, all while masquerading as another user. This allows him to use server resources. Further, protections like SYN cookies can be defeated since the attacker will now have the appropriate information to complete the connection. Thus, the SYN-flood utilization attacks of the nineties [Gar00] will become possible again.

This security flaw, resulting in lack of deployment, is one of the key reasons source routing is not usable for avoidance routing. However, source routing is also not viable for avoidance routing for several other reasons. Strict source routing requires the sender to specify *every* hop on a path. This requires topological knowledge, which we've already shown is difficult to get. If there is one error in the path specification, the data will not

reach the destination. Another major limitation of strict source routing is header and packet size. In IPv4, the source route path length is limited by the IPv4 header size. This results in a maximum path length of 8 hops, which siginificantly restricts the richness of available avoidance routes. This limitation in IPv6 is restricted to packet size, but the longer the source route, the smaller the data portion of the packet can be. A significantly complicated source route could result in very low bandwidth. Finally, strict source routing can override AS policy — something our avoidance routing system will not do.

While it is clear that strict source routing is not usable for avoidance routing, loose source routing also is not usable. We already identified a security flaw in loose source routing, but there is also a limitation in regards to avoidance routing specifically. Recall that loose source routing allows the sender to specify a list of stops, but how the packet goes from stop to stop is up to each router. Imagine that I want to send a packet from the United States to Germany, while avoiding France. I specify a route that goes from the US to Spain to Italy to Germany using loose source routing. However, I have no guarantees that in any of these routing steps, the path will not traverse France. I could reduce the likelihood of "strange" routing by decreasing the granularity of the path. In order to decrease the granularity, one needs better topological knowledge, thus returning us to the topology problem.

One could argue that an overlay network could be used to accomplish avoidance routing. That is, we strategically create an overlay network such that each node has complete annotated topology information. We simply use link state routing in the overlay to find a route that meets a user's criteria. Unfortunately, just like loose source routing, the underlay network is out of our control. Again, we could reduce this with finer granularity, which once again returns us to the topology problem. Further, decreasing granularity in a overlay network requires deploying more nodes, which makes the network increase in overall cost.

It should be clear from this discussion that neither source routing nor overlay networks are sufficient to meet the goals of avoidance routing. However, this discussion has high-

lighted several of the major challenges that avoidance routing must overcome. The foremost of them is creating a solution for the topology problem. Further, we must not repeat the mistakes of the past. Avoidance routing must be created in a manner such that it cannot be used as an attack vector. Finally, there are serious trust concerns that an avoidance routing solution must overcome. Each of these challenges will be discussed in more detail in the following sections.

2.2 Internet Topology

In order to have avoidance routing work, it is necessary to have a view of the Internet topology. We need this view to determine which routes match or violate the constraints given by the user. With an incomplete topology, it may not be possible to find a path from source to destination given the appropriate constraints, even if such a path exists.

Unfortunately, learning the topology is a well-known hard problem [OPW10]. This problem is created by the routing technologies and institutions currently used on the Internet. The first problem comes from the Border Gateway Protocol (BGP) [RLH06], which is the routing protocol that makes up the backbone of the Internet. In BGP, each router announces that it has a path to each of its neighbors. As a router receives an annoucement, it adds the neighbor it received the announcement from to the advertised path.

Let us use Figure 2.1 to illustrate how BGP works. Each node in this graph will advertise to its neighbors that it has a path to all its other neighbors. Thus, C will advertise to D and B that it has a path to E. Node C will also advertise to E that it has a path to D and B, and so on for every node. When B gets the advertisement from C specifying that it has a path to E, it will add C to the specified path, forming the path to E that is $C \Rightarrow E$. Node B will then advertise this new path to all its neighbors as well, telling both A and D about the C $\Rightarrow$ E path. In this way, every node will learn a path to every other node.

The important limitation occurs when a node has multiple paths to a destination. For

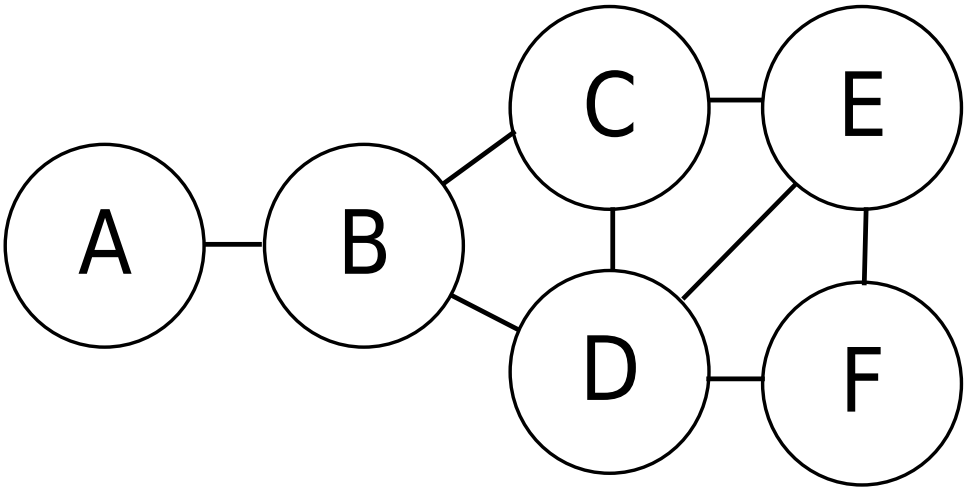

Figure 2.1: A simple six-node topology

example, B will eventually learn the C ⇒ E path as well as the D ⇒ E path. However, in BGP, you only advertise a *single* path to any destination. In this way, B will choose either the C ⇒ E path or the D ⇒ E path to advertise to A. This decision will be made through BGP policies, which we will discuss later.

As you may notice, node A only has one neighbor, B. Because of this, node A can only know one advertised path to every node. Let us assume that B chooses to advertise the following set of paths to A: {C, C ⇒ E, D, D ⇒ F}. Since A only has these paths to view the topology, it ends up with a perspective of the topology as illustrated in Figure 2.2. Notice that A does not know about the C - D link, D - E link or E - F link.

Now let us assume A wants to get to node E avoiding a property that C has, such as geopolitical location. Node A's view of the topology cannot let A construct a route that avoids C, even though such a route clearly exists using the D - E link. This example illustrates the partial view problem – any one node on the Internet only has a partial view of the entire topology.

This problem is actually exacerbated by BGP policies. How BGP decides which path to share is decided based on a policy, the most common of which is the no-valley policy

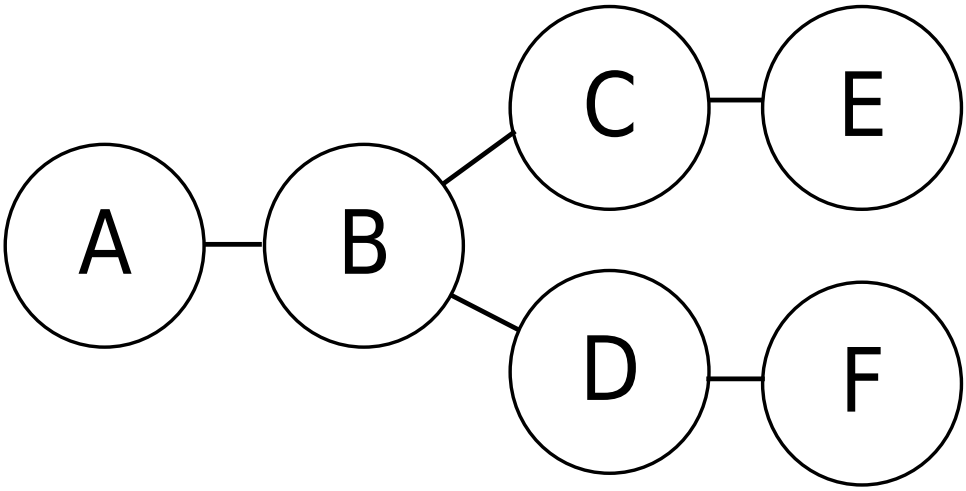

Figure 2.2: The simple topology as constructed from advertisements

[Gao01]. This policy is based on the fact that the Internet has a natural hierarchy derived from customer/provider relationships. The policy states that customers should not be used as transit routers.

In the Internet, there are a large number of low-degree edge nodes, that are customers of the intermediate nodes. These have higher interconnection and are, in turn, customers of the core nodes. In general, core nodes form a complete peering graph and are exclusively providers. The hierarchical structure is discussed in more detail in [SAR02] and [OPW08].

Figure 2.3 represents the hierarchical Internet graph. Here, Nodes S, O and D represent the low-degree edge nodes. These nodes are customers to nodes I-1 and I-2, which represent the intermediate nodes. I-1 and I-2 are both customers of the core node. As described, the core actually represents a small number of completely connected nodes.

Figure 2.3 is especially useful since it easily illustrates the no-valley policy. In the no-valley policy, you never go into a "valley" and always go over "mountains." Let us assume S wants to send data to D. There are two ways to reach D, either through the "valley" formed by O or over the "mountain" formed by Core. Since O is a customer of I-1, it cannot serve as a transit node for the data; thus we cannot use the "valley."

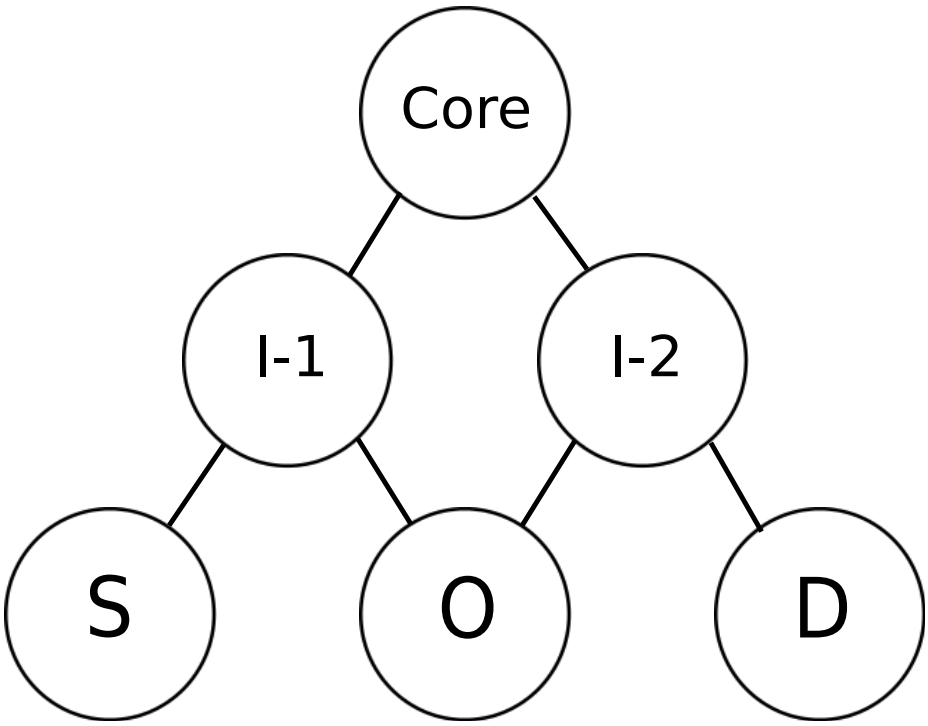

Figure 2.3: A topology showing the Internet hierarchy

This policy, as well as others, restricts the available routes advertised to any node. This means that, in general, the partial topology any node can view is even more limited since it is acted on by another set of topological constraints. Avoidance routing will need to overcome this limitation. We have made the case that constructing a topology from routing information is not possible. One could argue that we could construct a topology via other methods.

There are several methods that could be used, but they all tend to be prohibitively difficult. One mechanism is to use a tracerouting approach [SSK98] to build the topology. This strategy works by sending packets whose TTL expires after a certain number of hops. The router at which they expire will send back a ICMP response indicating the TTL failure. By incrementally increasing the TTL, we can learn which routers are connected to each other.

Unfortunately, this technique has a few problems that make it unusuable. First, all

packets are routed via advertised paths. It is impossible, using this technique, to determine unadvertised paths since no packet will ever traverse them. Further, routing changes and asymmetry may cause an incorrect view of the topology. For example, the packet may die at node X on route 1, but the packet may be routed on route 2 the second time, dying at node Y. We would then incorrectly assume X is connected to Y.

To overcome these problems, avoidance routing will be designed at the routing layer rather than at the end-to-end layer. Routers must be able to overcome the partial view problem in order to route to any destination, especially in the face of reliability concerns. By designing at the routing layer, avoidance routing will leverage the fact that while every node has only a partial view of the Internet, the union of all nodes, by definition, has a complete view of the Internet. In this way, avoidance routing will use a distributed decision to find paths. This will be discussed in more detail in Chapter 3.

2.2.1 Annotating the Topology

The difficulty of constructing an accurate Internet topology should be clear. However, this is only part of the topology problem that avoidance routing must overcome. Not only does avoidance routing require an Internet topology, but that topology must be annotated with security properties. Avoidance routing must know if a node meets or violates a given set of constraints. Without the appropriate annotations, it will be impossible to determine if a route is valid or not.

Annotating the graph is, unto itself, a hard problem. For example, how does one determine what country a router is in? This, partially, depends on who is asking. A neighbor who is physically connected to the router may be able to determine the answer via locality and proximity. An end user may try to use a triangulation technique to determine the router's location. Both of these answers may only work for location. What if we want to know the router's firmware or corporate ownership? Clearly locality or triangulation cannot answer those questions.

Determining how to annotate the topology is one of the core problems avoidance routing will attempt to solve. One point that should be clear is that every router knows its own security properties. If avoidance routing can simply leverage this fact, annotating the topology may become much simpler. In Chapter 3, we will discuss several mechanisms to help annotate the topology.

2.2.2 Disclosure Limitations

A critical problem that avoidance routing will face is information disclosure. Avoidance routing relies on making decisions based on different properties of routers. As previously stated, every router knows its own security properties. However, it is well known that ASes are secretive. They, in general, do not want to disclose information about their routers. In some cases, disclosing this information may be detrimental to their business or their router security.

Incentivizing information disclosure, as well as providing mechanisms for limited and partial disclosure, is a fundamental goal of avoidance routing. In Section 3.4.1, we will discuss the different types of properties that exist, showing that some properties *must* be disclosed for standard routing to work. We will also discuss incentives for additional information in Section 3.4.2, as well as mechanisms for partial disclosure in Section 3.5.1. Given all the techniques designed and implemented, we feel that avoidance routing will be able to effectively create and use an annotated topology.

2.3 Security, Scalability and Speed

When any network security solution is proposed, we need to consider the three "s" words — security, scalability and speed. It is imperative to consider security when proposing a security solution. As discussed in 2.1, source routing can be used as a denial-of-service attack vector, making it unsuitable for avoidance routing. Any solution used for avoidance

routing, in turn, must not become an attack on its own.

There are several security concerns that a solution must address. The first is to ensure that avoidance routing cannot be used as an attack vector against non-avoidance routing users. The second is to ensure that avoidance routing users cannot launch attacks against other avoidance routing users. Finally, we must ensure that avoidance routing cannot be used to attack routing infrastructure.

Any network system, in general, will have to deal with scalability concerns. The scale of Internet routing is already a known problem [ATL10]. The fact that each user of avoidance routing may want multiple different routes already suggests that scale may be a significant issue. One advantage of using a source routing solution is that scalability is not a problem. Each packet specifies its own route, thus each router only needs to follow the instructions. Unfortunately, as discussed, source routing is not an option.

Since avoidance routing operates at the routing layer, it will have to address the scalability concerns. How many simultaneous avoidance routes can exist is a critical question. Another important question is how many outstanding requests can be served at once, and how does an excess of requests affect other requests. Finally, similar to the security concerns, can the excess requests or routes negatively affect the existing ones?

Speed is the final S that is critical to network systems. What is speed? Speed, in routing terms, is how fast a router can push packets. For top tier ASes, the rate at which packets move is, in many cases, the most important concern. If avoidance routing is to ever be adopted, it **cannot** negatively affect the performance of standard packets. That is, no normal user should be able to tell if a router supports avoidance routing or not.

The optimal solution for avoidance routing allows both avoidance and non-avoidance packets to move at line speed. We have already stated that non-avoidance packets must not be affected, and thus will get preferrential treatment. In Section 4.2.1, we will discuss how avoidance routing can still achieve strong performance for avoidance packets. Further, in Section 5.8, we will evaluate the forwarding performance for both avoidance and non-

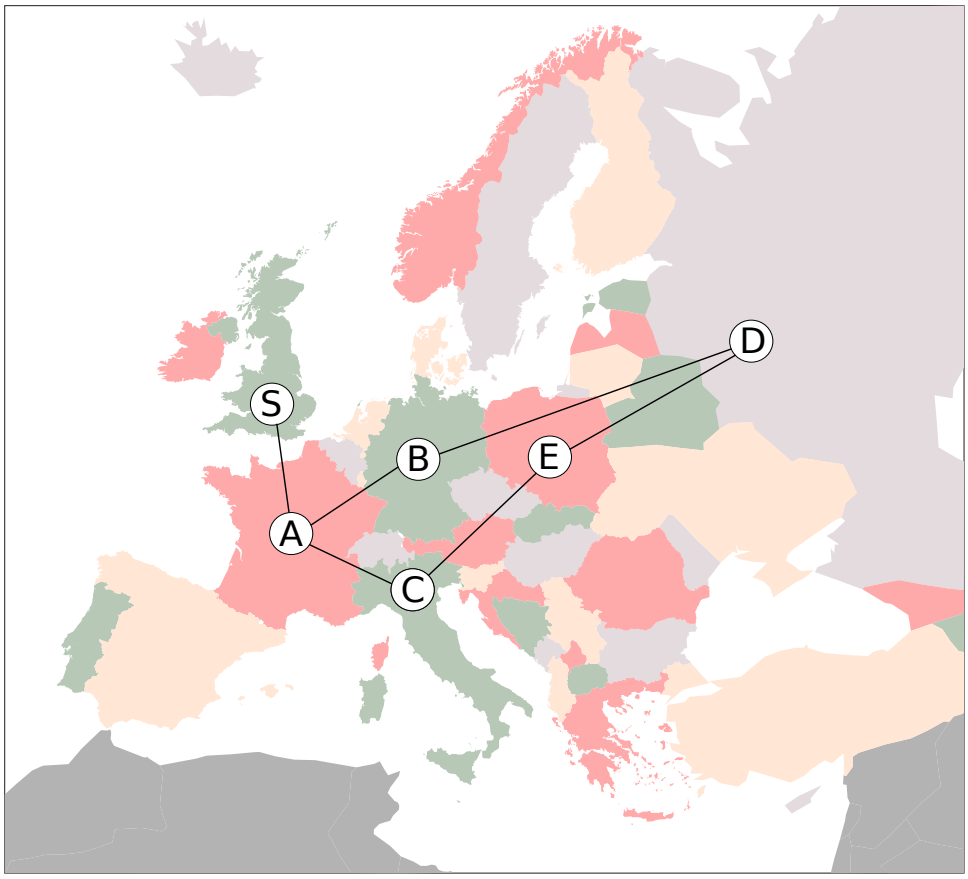

Figure 2.4: The original European topology

avoidance packets and show that avoidance routing does not negatively affect non-avoidance packets while still achieving excellent performance for avoidance packets.

2.4 Trust

Avoidance routing inherently relies on a lot of important information, such as the geopolitical location of routers. A very important question is how to trust the information we have? Some of this information may come from the questionable party themselves. Can we possibly trust this information? Further, individuals or conspiring groups may wish to falsify information such that avoidance routing does not work correctly.

x To make this more concrete, consider our original example, which is shown again in Figure 2.4. In our example, user S in the United Kingdom wants to speak to user D in Russia. For whatever reason, S wants to avoid any node in Germany. However, suppose node B can falsify its information, claiming that it is in the Netherlands rather than Germany. Because of this false information, A may forward to B rather than C, causing the information to explicitly violate these criteria.

A mechanism to protect and validate information is critical to ensure that avoidance routing works correctly. In Chapter 6, we will discuss the consensus mechanism that we use to ensure information is accurate. We will also discuss alternate mechanisms, and show why none of them will work to solve the information trust problem.

A second type of trust is also required for avoidance routing to work properly. Once again, refer to Figure 2.4. Instead of B lying about its properties, perhaps A simply ignores our wishes and forwards to B anyway. This may be a result of A colluding with B, or simply A disliking S. We need to be able to ensure that nodes properly route. While solutions may exist for this problem, we do not address this specific issue in this dissertation. Instead, we discuss the reasons why this is an extremely hard problem and look at some possible solutions in Section 7.1.

CHAPTER 3

Annotating the Topology

The goal of avoidance routing is to provide users with a mechanism for avoiding routers they do not trust. In Chapter 2 we discussed the challenges that avoidance routing must overcome to meet this goal. How our system meets the goals of avoidance routing while addressing these challenges is answered in the next several chapters. In this chapter we will discuss the fundamental problem of determining the Internet topology, as well as correctly annotating it with security properties. In Chapter 4 we show how the system actually performs routing while achieving strong performance. Finally, how we provide strong trust guarantees is argued in Chapter 6.

Avoidance routing allows users to specify security criteria that will determine how their data should be routed. Given this criteria, routers will route that data through nodes that satisfy the criteria. How does avoidance routing find a satisfactory route? Every user may have a different set of avoidance criteria. To find and store all routes with all possible instantiations of security properties is impractical, if not impossible. Therefore, we must design an *on-demand* system. This means that the system will find an appropriate route when the route is needed, and not store that route permanently.

Routing on-demand still requires an increased knowledge of the topology. In order to conduct routing that avoids nodes with specific security properties, the system must have an annotated topology, although not necessarily completely. Clearly, regardless of how we actually perform the routing, the system must be able to determine which routes are valid and which are not. This can be done by having complete topology knowledge and running

graph algorithms to determine the correct route. This can also be done by only having knowledge about one's neighbors and doing a blind search throughout the network. The approach we take is discussed in Chapter 4; however, it is clear that we need a topology.

We designed the system to work with BGP, since we assume that participating routers are already running BGP, and we can make use of the information they have already gathered about available routes. However, several additional steps are required. First, we must determine the set of possible security properties. Second, the security properties of each router and autonomous system must be determined. Third, the properties of advertised routes must be disseminated between routers. Given these advertisements, we can build an annotated topology.

Our discussion of the avoidance routing design will focus on how these additional steps are achieved. We will answer the following questions in this chapter. What are security properties? How are they determined? How does each router learn the security properties of other routers? Are routers incentivized to learn or disclose more information? These questions are answered in Sections 3.2 through 3.5.

3.1 Obtaining a Topology

. The principle reason that source routing or overlay networks are not viable solutions for avoidance routing is that they require knowledge of the Internet topology. Determining the current Internet topology is an open and hard problem. However, to successfully build a path from source to destination, some knowledge of the topology is required. Routing protocols solve this problem by propagating some information about the topology to every routing node. This information ranges from data concerning the neighbors of every node in the graph to the distance from any node to all of its neighbors. The former is common in link-state protocols and results in learning the complete topology of the network. The latter is common in distance-vector protocols and simply informs a node of how far a destination

node is from all its neighbors.

The most common inter-AS protocol on the Internet is BGP, which is a path-vector protocol. In this protocol, each node learns of paths to possible destinations from each of its neighbors. In this way, each node will know some of the topology of the graph while also learning the necessary distance metrics required for routing. Clearly, each node does not have a complete topology. Without a complete topology, how can avoidance routing determine a valid path to the destination?

The heirarchy of the Internet splits routing into inter-domain and intra-domain routing (from here on refered to as inter-AS and intra-AS). Intra-AS routing is generally covered by link-state protocols and allows routers within an AS to know the entire internal structure. Leveraging this information for avoidance routing is trivial. How avoidance routing operates for inter-AS routing is far more important. To perform avoidance routing, we will need knowledge of the AS topology, and avoidance routing will make AS-level, not router-level, decisions.

The advertisements in BGP result in each router having a partial view of the Internet AS topology. To accomplish the goal of avoidance routing without building a complete topology, we leverage the fact that the union of the partial views results in a complete view of the topology. In fact, only a subset of nodes are required to build a complete topology.

Definition 1. *A graph G is defined as E, edges, and V, vertices or nodes.*

Definition 2. *A graph G is connected if a pairwise path, p_{xy}, exists $\forall x, y \in V$.*

Definition 3. *An advertisement A lists all nodes along a path to any given destination from the advertising neighbor, including the neighbor.*

Axiom 1. *In BGP, each node has at least one advertisement from all neighbors.*

Theorem 1. *Let the Internet AS graph, I, be a connected graph. Let V be the set of all nodes in I. $\exists X \subseteq V$ such that $\cup A_X = I$, where A_X are the advertisements stored in X.*

Proof. Ignoring all other information, each A_{Xi} specifies a neighbor, n, of i, where $i \in X$.
All neighbors of i are equivalent to E_i, the edges out of i.
If $X = V$, then $\cup A_X$ specifies all nodes V and all edges E.
Thus $\cup A_X = I$ for $X = V$.
Let $X = V \setminus \{v\}$.
A_X specifies all nodes $V \setminus \{v\}$.
If I is connected, then v is specified in some ad A_x by v's neighbor(s).
All edges out of v, E_v are specified in some ad A_x by v's neighbor(s). Therefore $\cup A_X$ specifies all nodes V and all edges E.
Thus $\cup A_X = I$ for $X \subset V$.

□

Theorem 1 states that there exists a subset of ASes on the Internet such that the advertisements of these nodes describe the entire Internet AS graph. The first part of the proof proves, trivially, that this is true if the set includes all the nodes on the Internet. The second part shows that if the Internet is connected, then this is true even if the set does not include all the ASes on the Internet. This is true because a missing AS will necessarily be described by its neighbors' advertisements. Theorem 2 shows that the nodes required to describe the Internet are strictly smaller than the Internet, regardless of connectivity, unless all nodes are disconnected.

Theorem 2. *Let the Internet I be a graph where $E \neq \{\}$ and I is not fully-connected. Let V be the set of all nodes in I. $\exists X \subset V$ such that $\cup A_X = I$, where A_X are the advertisements in X.*

Proof. Let $X = V \setminus Y$ where $\forall y \in Y$, y is connected to X and y is not connected to any other nodes in Y.
A_X specifies all nodes $V \setminus Y$ and $\forall y \in Y$, y is specified by some ad A_x since y is connected to X.

A_X specifies all edges out of Y and all edges within X. Therefore $\cup A_X$ specifies all nodes V and all edges E.
Thus $\cup A_X = I$ for $X \subset V$.

□

Basically, this theorem states that any connected graph (except a fully connected graph) can be specified by the advertisements of a subset of the graph. This proof shows that this subset is any subgraph where the removed nodes do not have a link between them. Further, a disconnected graph can be specified by less than the entire graph as long as all strictly disconnected nodes are in the subset and there exists at least one edge. For the case of avoidance routing, a disconnected graph is not useful. It is reasonable to assume that the Internet is a connected graph and not a fully connected graph.

The proofs conducted so far do not make use of any additional information in the advertisements. These proofs only leverage the fact that the advertisements carry neighbor information. The proof for Theorem 2 is limited because it requires that the nodes not included in X (Y) are not connected. With only neighbor information, you cannot determine this hidden link. However, information about additional links can be determined through the extra information in the advertisement. It can be shown that the nodes in Y can be connected as long as the connection is propagated in an advertisement in A_x.

These theorems show we can build a complete AS topology without any one node having complete topological information. Further, we can build this topology without having to visit every node. In Chapter 4, we demonstrate how we leverage this information to properly route by having each node use its partial view to make a correct routing decision. However, having proven that we can construct a topology through BGP advertisements, we must now discuss how we annotate it. The rest of this chapter will answer this question.

3.2 Security Properties

Avoidance routing allows users to avoid routers with properties they do not trust. Given this statement, an obvious question is what are these properties? Security properties are attributes intrinsic to each router. For example, a security property might include geopolitical location of the router. We have not determined the entire list of possible security properties as we have made the design choice to leave this open. We wish to leave it open so that future developers can add new security properties to the system that we did not perceive. Thus, we will design a system for a variable number of security properties. A method by which developers can agree upon future security properties will probably be required, but we feel that this is outside the scope of our research.

While we do not know the complete set of properties, there is a limitation on the kind of properties we allow. Currently, avoidance routing only supports path-independent properties. These are properties which do not change based on the route taken to a node. For example, geopolitical location is path-independent. Regardless of how I get to a router, that router will always be at the same location. The other class of properties is path-dependent properties. These properties change based on the route taken to a node. For example, in quality-of-service, total delay may be an interesting property. However, the path taken to any node changes the current total delay. Arriving at a node with delay X does not mean that any path to that node has delay X. These properties greatly increase the complexity of path selection and path reusability, and thus we have removed them for performance.

We have identified a few security properties that we believe will be useful for the system. These properties are geopolitical location, corporate ownership, firmware version and manufacturer, autonomous system (AS) number, and reputation level. We also believe that a more theoretical property, path attribution level, will be interesting to consider and show how the system can evolve to future technologies. Further, we will discuss two properties

that come into play based on our consensus system. This system will be discussed in great deal in Chapter 6. We will briefly discuss why we think each of the identified security properties is useful.

3.2.1 Geopolitical Location

Geopolitical location can be considered a necessary requirement for the system. Many nations and individuals do not trust specific countries, and nations or alliances do not want critical intelligence to be available to their adversaries. For routing done on behalf of national governments, ensuring that data does not transit adversarial nations is critical. Further, some nations have legal restrictions on what data one can possess. For example, encryption restrictions exist in many countries [RSA12]. Dissidents or other individuals may have specific reasons to avoid certain countries. These reasons force geopolitical location to be an obvious and necessary security property for the system.

Intelligence may not be the only goal of an adversarial country. A country can selectively drop or poison packets to deny service to opponents. Some countries will actively prevent dissident or provocative information from transiting their borders. Finally, some countries have laws that are more relaxed on what their citizens can do on the Internet. Several well-known botnets and criminal organizations originate in specific countries. Avoiding these countries altogether may greatly increase the security of important data.

A secondary reason for geopolitical location is that it provides a feasibility analysis for the system. Geopolitical location is probably the coarsest granularity by which routers can be meaningfully grouped. If we cannot actually perform avoidance routing at this granularity, it seems foolish to attempt to do it at a finer granularity. This second reason emphasizes the need to include geopolitical location as a security property, but also highlights the need to do an extensive evaluation of the system by way of a realistic Internet map that uses this property.

3.2.2 Corporate Ownership

Corporate ownership is another obvious security property. Recent cases of corporate espionage show us that many corporations distrust other corporations. Further, specific users may have a grudge or distrust of corporations based on past experience or past employment. Corporate ownership also is another property relatively easy to test. Services like Whois [Dai04] allow us to see a mapping of IP address to corporate owner. While this mapping may not be entirely accurate, it gives a realistic picture on which to test the system. These criteria make corporate ownership an excellent choice as a security property.

3.2.3 Firmware Version and Hardware Manufacturer

Every day we learn of new security vulnerabilities in software X or hardware Y. Routers are no exception to this rule. Because of this, individuals may wish to protect their data by avoiding specific firmware versions or hardware manufacturers with a known vulnerability. Avoiding all routers in France may not be helpful if the French have compromised other routers. Picking paths that use routers least likely to be exploited will help increase security. In a specific case, certain hardware manufacturers may not want their data to transit the of their competitors.

3.2.4 Autonomous System

As we discussed with geopolitical location, it is well known that certain ASes on the Internet are less than scrupulous. These ASes may selectively interfere with traffic or host nefarious businesses. Further, some ASes may be more cooperative with government filtering and censorship programs. Because of this, users may not want their data to transit or reach these ASes. This clearly qualifies AS number as a security property. Further, the AS number is easily available to every border router because it is passed along in the path advertisement in the Border Gateway Protocol (BGP). This is useful since BGP is the

primary routing protocol of the Internet.

3.2.5 Reputation

Reputation level is a unique security property. It allows a third party to rate each AS or router with a reputation level. How that reputation is determined is up to this third party. However, this third party could be used to determine if ASes are properly performing avoidance routing, or are lying about their security properties. Depending on how ASes behave, their reputation for being honest can increase or decrease.

3.2.6 Path Attribution

Path attribution is a unique security property. Path attribution is the ability of a router to verify the source and destination of a packet, as well as the nodes it's traversed. However, currently no decent path attribution systems exist, but there have been some recent efforts [GBH09] that explore attribution.

Having an attribution system would work directly in conjunction with avoidance routing. Using avoidance routing allows attribution systems to have stronger guarantees. One can choose to only use routers that will provide strong attribution information. By avoiding routers that do not support attribution (or strong attribution), both sender and receiver can have a more solid belief in whom they are talking to. Because of this, path attribution offers a way for us to show the extensibility of the system for future technologies. For this reason alone, we feel it should be included as a security property.

3.2.7 Consensus and Agreement

A critical problem for avoidance routing is determining how to trust information. We discuss how we solve this problem in Chapter 6. Our approach revolves around determining what the consensus opinion is about certain information. Whether a router is in the

consensus (*in agreement*) or not may be used as a security property. Further, how often a router is in or out of agreement may be a security property. Both of these provide a user with a mechanism for stronger assurances that their constraints are satisfied. This will be discussed in far greater detail in Chapter 6.

3.3 Property Changes

Security properties are the core unit of avoidance routing. Routing decisions are made based on the value of these properties. How often these properties change may affect whether avoidance routing is properly used. Further, how a router learns about property changes is important. Many of the properties we've described are relatively static, such as geopolitical location or corporate ownership. It is reasonable to expect that routers are not moving or changing hands often. Other properties, such as hardware or software, may change more frequently. However, deploying new hardware is not something a company can do on a daily basis, and many router operators are reluctant to deploy new software until they are certain it will not harm their bottom line. For these reasons, we do not expect these properties to change often.

It is possible that our consensus mechanism will change its values often. As we will describe later, the consensus mechanism is calculated locally, and so any changes to its value will be "instantly" determined by the forwarding router. Thus, we believe that the majority of our properties will change infrequently; less often than routing changes. Further, changing many of our properties will explicitally lead to routing changes (deploying new cables, software or hardware). While there may be a period of incorrect routing, we feel this period will be no greater than a similar period that occurs during routing changes.

3.4 Determining Security Properties of Routers

The question of what security properties are has been answered. This leads us to the next question: How do we determine what security properties a router has? Further, who determines these properties? While it may be possible for every router to determine the security properties of every other router, we believe this is infeasible. If this information was easily available, it would make more sense for end hosts to obtain it and use source routing, rather than gather it at each individual router. Even with this information, we would still have to solve the general topology problem. Having global knowledge of security properties would only help an AS determine if a route intersects specific properties, but not if an alternative avoiding route exists.

An Internet-wide link state routing protocol could make use of this global knowledge to do avoidance routing. However, the routing community seems hesitant to make fundamental changes to Internet routing. Further, ASes do not want everyone else to have knowledge of their connectivity. Finally, it is not clear that a Internet-wide link state system would be able to scale and have adequate throughput. Because of these reasons, as well as those above, we decided to choose an augmented BGP system. Other possible alternatives are discussed in greater detail in Chapter 8.

Since we cannot gain global knowledge of security properties, the properties of routes must be determined in another manner. Obviously, every AS is aware of the security properties of its own routers. However, we cannot necessarily rely on ASes to be either cooperative or truthful about this information. On the contrary, it may be in the best self-interest of an adversary to lie about this information. In order to increase the difficulty of lying, each node is responsible for determining the security properties of its neighbors.

How does a node determine the security properties of its neighbors? This greatly depends on the security property. For example, an AS number may be trivial to determine through Whois databases and peering relationships. Other information, such as router

manufacturer, may be more difficult to determine. To simplify the problem, we have identified four classes of properties. Each class has different levels of disclosure and ease of determination.

3.4.1 Property Classes

There are four classes for which all security properties fall into. The first class includes properties that an AS **must** disclose to its neighbors for routing to properly function. AS number is the exemplar of this class. In BGP, the AS number operates as a unique identifier for every AS on a route. BGP uses these AS numbers to determine if routing loops or other routing failures have occurred. Thus, ASes are required to disclose their AS number. Further, technologies like S-BGP [KLS00] are designed to prevent ASes from falsifying their AS number.

Properties that an AS is **willing** to disclose to its neighbors are in the second class. This generally includes information that is already disclosed for business reasons. For example, corporate ownership is disclosed to all peering neighbors during negotiation. Each side needs to know who to pay money to, or expect money from. Other information in this category may include geopolitical location. Who lays the connecting cables and where they go may need to be negotiated in the contract. In many cases, peering routers live in the same building.

The third class of properties includes those that an AS is **unwilling** to disclose to its neighbors, but can be **learned** by a neighbor. An example of this may be router manufacturer or firmware version. Certain types of hardware or software are known to have artifacts — signatures that identify them to an observer. A neighbor may be able to observe these artifacts and determine information about that neighbor. Geopolitical location may also fall into this category. An AS may not be willing to disclose the location of its routers, possibly for security concerns. However, a third party may be able to determine the geopolitical location of all routers using technical means, such as triangulation, and

make this available via a service similar to Whois.

The fourth and final class of properties are those which an AS is **unwilling** to share and **cannot be learned**. A hypothetical example of this would be how much a business paid for a router. There is no technical means of learning the price paid for equipment without obtaining the contract. Properties that fall into this category are unusable for avoidance routing. A property that is undisclosed and cannot be learned is equivalent to a property that does not exist.

3.4.2 Incentivizing Disclosure

The four classes of properties describe how it may be possible to determine the properties of routers. Clearly, some properties exist in the first and second classes. This means that avoidance routing is viable without requiring further disclosure. There are probably a significant number of properties that live in the third class. These properties are usable for avoidance routing but clearly will have varying degrees of accuracy depending on the technique used to obtain them.

It would better if ASes could be incentivized to move property information from the third and fourth classes to the second class. A possible but poor option is to make these properties required for Internet routing to work correctly. It has been shown that geolocation information can help make better routing decisions while saving on storage cost [OLZ07]. However, fundamental changes to the routing infrastructure does not happen quickly; thus this strategy is not the best.

ASes may be willing to move information from the third to second class on their own. This could occur if several third parties begin to collect this information. Rather than being at the risk of third parties propagating false information, ASes can control the accuracy of information by disclosing it themselves. Further, corporate interests may create a market for avoidance routing, facilitating disclosure. A company that does not disclose may find itself left behind in potential future markets.

A final disclosure mechanism is through government mandates. We expect that governments will be a likely first adopter of avoidance routing. Their desire to use avoidance routing may cause them to mandate disclosure. For example, if NATO countries all wish to use avoidance routing, they may mandate the disclosure of geopolitical location. As more routers become avoidance routing capable, the interest in using the system may also increase; this may result in a snowball effect. As the number of people caring about disclosure increases, ASes may be forced to capitulate.

3.5 Dissemination of Security Information

Once the security properties of each AS have been determined, the ASes must determine the security properties of specific routes. As stated in Section 3.4, it is infeasible for each router to determine the security properties of every other router. Instead, each router shares security information with its neighbors. Routes containing security properties can be built up and shared in the standard distance vector or path vector manner. Using this approach, we can leverage BGP to help pass information about the security properties of each AS in an advertised path (by using BGP optional path attributes, for example).

The technique we use is similar to the augmented addressing scheme of Oliveira et al. [OLZ07]. In their scheme, they augment BGP advertisements such that geolocation is encoded along with AS number. They find that using geolocation information can determine better routes, rather than simply using the smallest hop count. They also find that this allows for better aggregation, as you can now aggregate based on location (for example, aggregating all destinations in Los Angeles into a single or a small number of entries, which leads to smaller routing tables).

While their scheme is designed to solve a different problem, we believe the augmented advertisement technique can be applied to our system. In avoidance routing, advertisements are augmented with all the security properties of an AS. We determined that the best

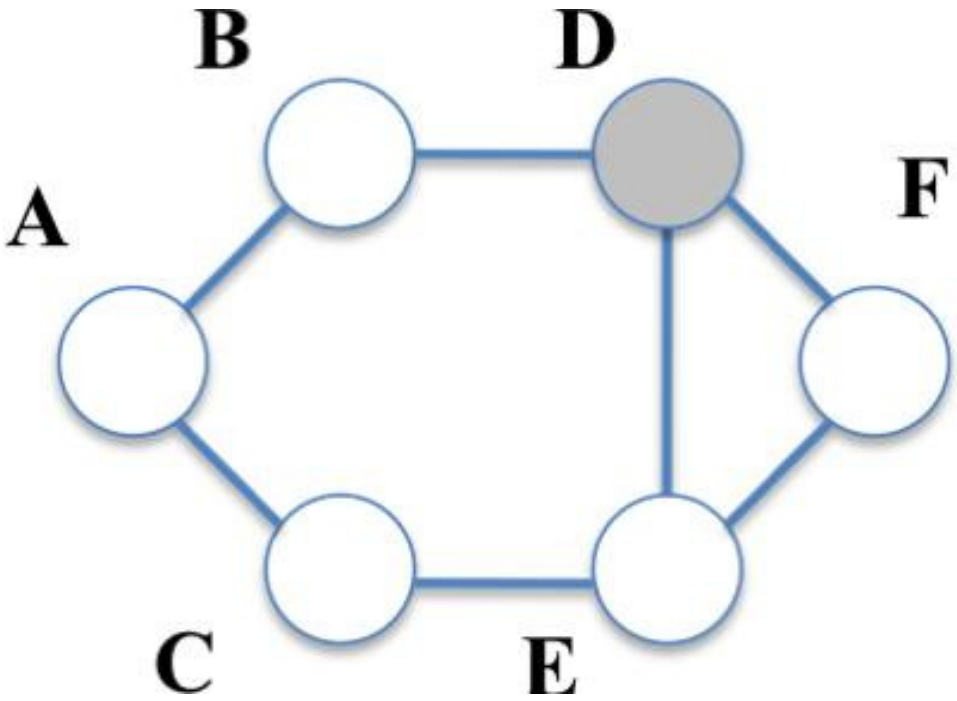

Figure 3.1: A simple network topology

location for this data is in the BGP optional path attributes, allowing legacy BGP speakers to operate without understanding avoidance routing.

We treat security properties like any other routing dynamic. If a security property changes, new BGP updates must be distributed. However, avoidance routing must not be a burden on BGP. We believe that security properties will change no faster than routing changes; therefore, requiring BGP updates for changes seems reasonable. This assumption is reasonable because changing security properties generally requires manual intervention, such as deploying a new router, and thus occurs infrequently. Even properties that can be changed automatically, e.g., upgrading firmware, occur on the order of months, not minutes. Further, many changes in security properties would already result in routing changes. For example, if an AS deploys a new router in a different country, this will undoubtedly generate routing changes along with the change in security properties.

The list of path security properties is called the *security vector*. The security vector is

built up step by step as the path is created, just like the normal BGP path vector. For example, consider the route from A to C to E to F in Figure 3.1. This route originated when E advertised to C that E has a direct route to F. Since each AS only has one security property in this example, we shall denote the security property of F as *f*, E as *e*, etc. E's advertisement to C then contains the security vector *f*. C then advertises an E-F route to A, including the security vector $e - f$. A will finally store the C-E-F path vector with the security vector $c - e - f$.

It is important to realize that the security vector has a one-to-one correspondence to the path vector. That is, C has the security property *c*, E has the security property *e*, etc. If F had two different security properties (e.g., located in two countries), it would specify both. In this case, the path vector would be C-E-F with security vector $c - e - f_1,f_2$.

Even though neighbors are responsible for adding properties to the security vector, they may still lie. In order to prevent lying or error, avoidance routing uses a consensus system that is discussed in Chapter 6. The important detail here is that the consensus system requires each AS to have as much information as possible from its neighbors. In Figure 3.1, AS A only knows the security properties of B that it already knew. The security vector does not tell A any more information about B. In order for A to learn D's information about B, D would have to advertise a path to B that reaches A. In some cases, this may never happen, especially given the hierarchical nature of the Internet.

To solve this problem, the advertisement contains a second set of information. When a node gives an advertisement to a neighbor, it adds the security properties about that neighbor to the advertisement. This can be viewed as a second vector called the *forward security vector* (FSV). Again, consider the example in Figure 3.1. When F advertises destination F to E, it adds *e* to the FSV. E will add *c* to the FSV when it passes the advertisement to C, resulting in a FSV of $c - e$. A will receive and store an advertisement to F with the path vector C-E-F, a security vector of $c - e - f$ and a FSV of $a - c - e$. How the security vector and FSV are used for routing, as well as accuracy calculation, is

discussed in the next several chapters.

3.5.1 Partial Disclosure

Some ASes may wish to participate in more limited forms of avoidance routing. These ASes may be willing to distribute some of their security properties, but not others. We allow this capability by making the security vector a variable length. In this way, an AS can add to the vector only the security properties it wishes to share. If an AS does not provide a specific security property that a user cares about, this AS is treated as untrusted in regard to that property.

Even with the variable disclosure, an AS may not want any of its information to propagate through the network. We have created a version of avoidance routing that we call *neighbor-only avoidance routing*. This form of avoidance routing operates only on the minimal set of information required to properly route. In the beginning of this chapter we mentioned that avoidance routing can be done as long as you have information about your neighbors. This is the minimum information required to run avoidance routing. A router must know if a neighbor is valid or invalid given a set of criteria, and make decisions accordingly. This type of avoidance routing minimizes disclosure while also harming performance. To understand this trade-off, we also implemented this version of avoidance routing. How this version works and how well it performs are discussed in the next chapter.

3.5.2 Advertisement Overhead

How avoidance routing propagates security properties has been discussed at a high level. Avoidance routing increases the size of BGP advertisements. What is the overhead cost for BGP advertisements with these modifications? If the changes are too costly, that is BGP advertisements become huge, avoidance routing may be unusable. To answer this question, we need to have a slightly better understanding of how avoidance routing works in detail.

Each security property has two pieces of information — the property type and the property value. The property type is the broad category of property, such as geopolitical location or router manufacturer. The property value is the exact property instantiation. For example, if an AS is located in Italy, it will have a property of type "geopolitical location" and value "Italy." For processing purposes, we map semantic types and values to numerical types and values. So the type "geopolitical location" corresponds to type 1, and the value "Italy" could correspond to value 22.

Why have both type and value? Why not just have each possible value be unique? So "Italy" would simply have value 22 and "Cisco" would have value 23. The main reason to separate value and type is for quick processing. Remember that an AS may only have information about certain types. If a host cares about a type that is unknown, the AS is treated as having all values for that type. But if there are no types and only values, the entire security vector would have to be checked for all possible values of that type.

For further speed of processing, we sort the entries of the security vector by type and value. Each security vector entry is a sorted list of sorted lists. The inner list is all the values of a single type, sorted by value. The outer list is sorted by type. For example, for the path vector X, Y, we could have the security vector (1 (1 3)) (2 (4 6)) (4 (3 8)), (2 (3 4)) (4 (3 5)). In this case, X has the properties (1 1) (1 3) (2 4) (2 6) (4 3) and (4 8) where the first number is the type and the second number is the value. Y has the properties (2 3) (2 4) (4 3) and (4 5). Since we group by type, we only need to list the type for the list and not for each individual property.

As discussed in Section 3.5, each AS makes claims about its previous and next neighbor on the path. Clearly, another neighbor may have already made the same claim. Figure 3.2 shows a simple example of redundant claims. It is likely that two neighbors of the same node may have the same information. Thus, the forward claims about a node may equal the backwards claims about the same node. In this example, notice that A and C are in agreement about B, as well as B and D about C. To save space in advertisements, a node

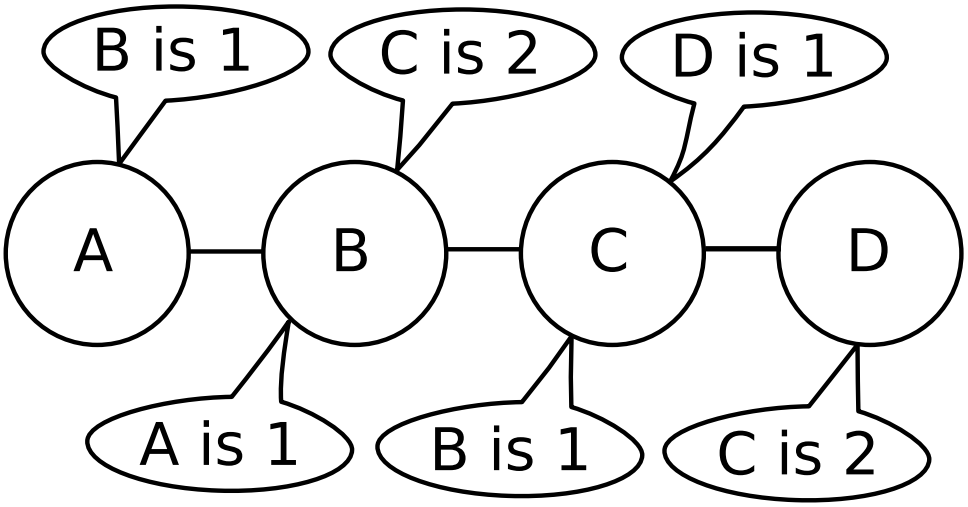

Figure 3.2: A simple topology showing bidirectional claims

can indicate its agreement with another node's claims rather than redundantly placing these claims. If there is disagreement, a node is allowed to specify only the differences, rather than all the parts it may agree with.

The final piece of the overhead puzzle is the size of a type and value. Currently, each type is 16 bits, giving a possible 65,536 different property types. We do not know how many avoidance types may actually exist, so the number of possible types is left large to allow for future growth. If it turns out that the number of types is small, a future avoidance routing implementation may more efficiently use space.

The size of values are defined by the governing type. Geopolitical location may only use 10 bits. Since there are currently only 200 countries, this may leave adequate room for future growth. Further, values could be reused if a country becomes defunct. On the flip side, router firmware version may have many possible instantiations, requiring far more bits. For this evaluation, we assume each value is 32 bits, which may be an overestimate.

Armed with the details, we can now understand how a BGP advertisement grows. Clearly, an advertisement grows with each new hop. It also grows every time a new type is added to the list or a new value is added for a type.

Table 3.1: Growth of advertisement with or without agreement

	With Agreement	Without Agreement
Per Hop	12 bytes	16 bytes
Per Value	4 bytes	8 bytes
Per Type	8 bytes	16 bytes

Table 3.1 shows the increase in advertisement size per-hop, per security property value and per security property type. We assume that when a new type is added to the vector, a value is also added. The per hop increase assumes that each hop adds one type and thus one value. For example, if we look at an AS path of 30 hops, 5 types with 2 values per type, we end up with an increase in ad size of 1864 bytes.

According to the Team Cymru Internet Monitor [Cym12a], the average AS path length is 4.61 ASes. Further, the longest AS path without prepending is 30 ASes.This means our example is a rough estimate of an upper bound for path length. Since we cannot know how many types of security properties may end up actually being used, it is difficult to estimate a true upper bound.

CHAPTER 4

Routing Using Security Properties

Up to this point, we have not discussed routing at all. In the previous chapter, we discussed the information required to make avoidance routing function. We showed how we use that information to build an annotated topology, as well as how we propagate that information throughout the network. We will now discuss how avoidance routing discovers a valid route to the destination and how it makes use of that route.

Avoidance routing cannot function as a standard "routing table" mechanism, where a path to every destination is stored for quick look-up. It is possible that there are a vast number of possible criteria users may care about. To store all paths with all possible criteria to all destinations would be impractical, if not impossible.

Instead, avoidance routing will operate in an on-demand fashion. A valid path to the destination will only be found when a user requests it, given a set of criteria. How the user makes the request, as well as how the path is built, will be explored in this chapter. We will also evaluate the search mechanism, showing that the search occurs relatively quickly. Further we will show that non-avoidance packets are not affected by avoidance routing, and avoidance packets incur no penalty once the path has been constructed.

4.1 Avoidance Requests

Avoidance routing begins when a user requests use of the service, and specifies the destination and security criteria he wants to avoid. The user's system will then create an *avoidance*

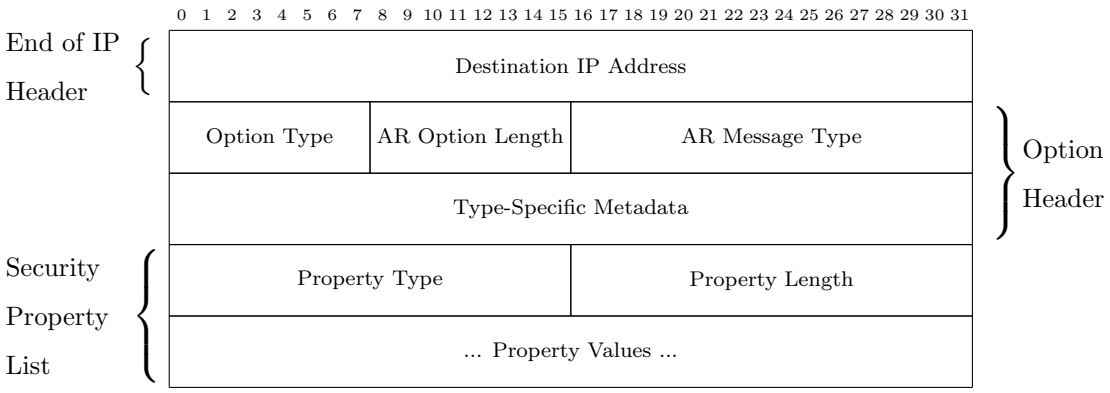

Figure 4.1: The header form layout

request, a message that contains the destination and criteria. This request will be given to the first hop of that end-user, which will begin to construct the path. In avoidance routing, the first hop may be the user's system itself.

The structure of this message can take two forms. The first form is the *header form*. In this form, the avoidance criteria is placed as an IP option. This option specifies the *avoidance criteria*, the properties the user wishes to avoid. The header form is designed for those who only have a small number of criteria. We assume that if the number of criteria is small, finding a path may be fairly quick. In this way, header form can be placed on top of a standard transport packet.

Since IPv4 headers have a maximum size, we have also created the *packet form*. This form places the avoidance criteria as the payload of the packet. The packet form allows a user to specify far more criteria then the header form can. In IPv6, we only use the header form since IPv6 options are extension headers and are not limited by header size.

In both forms, the destination desired by the end-user is the destination of the packet. Figure 4.1 shows the format of the header form. The header form consists of two parts, a preamble and payload data. The preamble consists of the option type, option length, message type and type data. The preamble is designed to allow a router to determine what the avoidance routing option is trying to say. All IP options begin with the option type so

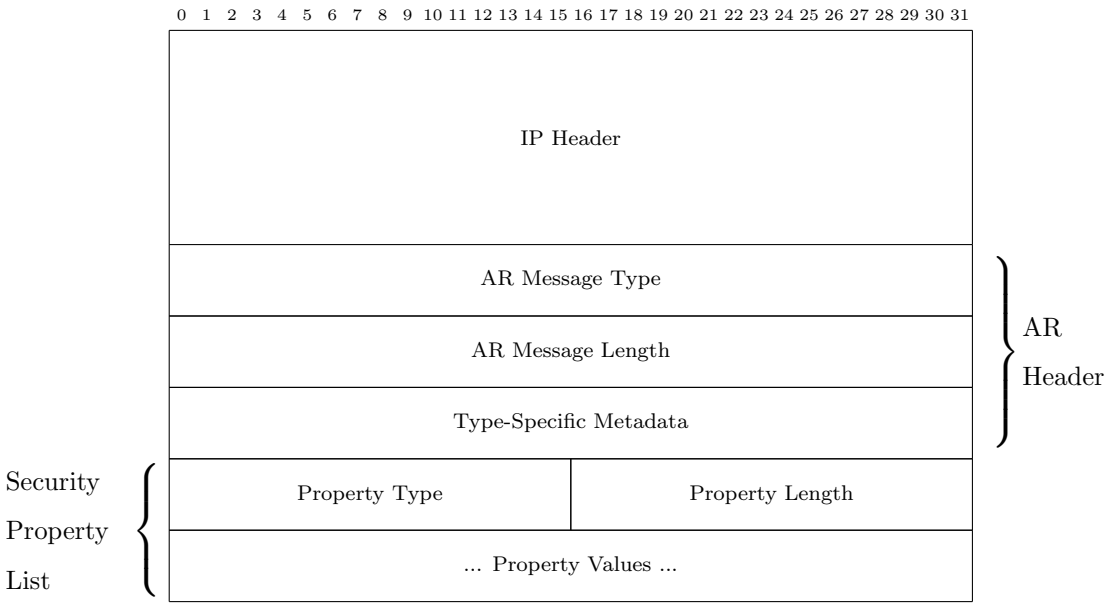

Figure 4.2: The packet form layout

a router can determine what the option is. Since IPv4 headers are limited in size, only 8 bits are required to specify the option length. Finally, the preamble specifies the avoidance routing message type and type-specific metadata. In this case, the message type would be an avoidance request. The message type metadata will be discussed in later sections as it is used to relay information between routers.

After the preamble, we place the payload data. In an avoidance request, it is the criteria to be avoided. This is a sorted list of property types and values, similar to that in the BGP advertisement discussed in Section 3.5.2. In this case, the property type is listed followed by a length for the number of values specified for this type (length is in bytes). Finally, all the values are listed, before a new property type is listed. Due to the length limitations of IPv4 headers, only 40 bytes of option space are available. With 8 bytes in use by the preamble, only 32 bytes are available for the criteria. This limits the header form to 6 property types, each with a single 1-byte value, and 1 property type with 28 bytes for values, or some place in between.

Figure 4.2 shows the packet form of an avoidance request. This form shares a similar

structure to the header form. First, the IP header must specify the avoidance routing protocol in the protocol field. The avoidance routing header then specifies the avoidance routing message type, length and metadata. Since space is not nearly as much of a concern as in the header form, the type and length are both 4 bytes. Again, the message type would be an avoidance request. Finally, the security properties are listed out in the same manner as the header form. This allows for relatively quick processing in both the header and packet forms.

In comparison with the header form, the packet form can specify far more properties. Most modern networks use an MTU of 1500 bytes. Given 20 bytes of IP header and 12 bytes of avoidance routing header, there are 1468 bytes available for criteria. This could specify one property type with 366 4-byte values, 183 different property types with one 4-byte value each, or somewhere in-between. If the user is not concerned with IP fragmentation, there are 65504 bytes available for criteria.

It may be possible to save space by defining aggregate properties. One way to aggregate is by similarity, such as aggregating all of Europe into a "Europe" property and stating "I wish to avoid Europe." Another mechanism may be to aggregate based on usage. A large percentage of users may wish avoid the US, China and Cisco routers. Creating a aggregate category that avoids all of these may be useful in saving space for requests. Routers would have to understand how to deaggregate a property in order to make actual avoidance routing decisions.

As stated, since IPv6 headers are not limited in size, there only exists the header form in IPv6. However, IPv6 options operate slightly differently than IPv4 options. Figure 4.3 shows the IPv6 header. The primary difference is that the option type is specified in the next header field of the *previous header*. Further, the next header field in an option will specify the type of the subsequent header. Otherwise, the option is structurally the same as the IPv4 option.

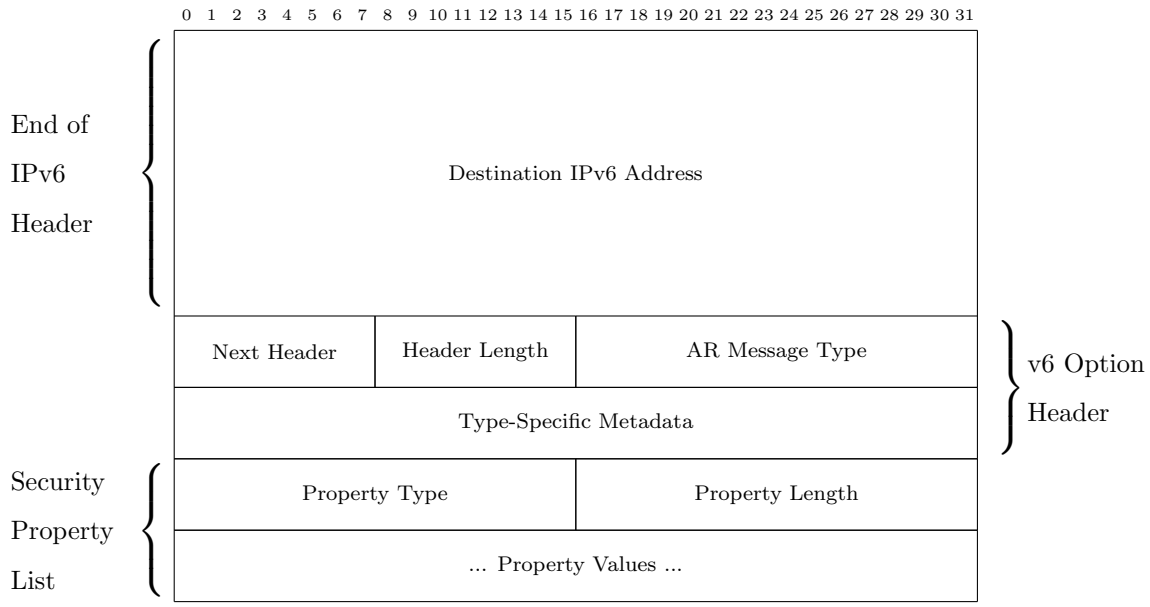

Figure 4.3: The IPv6 header form layout

4.2 Depth-First Search

It is time to ask the fundamental question of avoidance routing. Given an avoidance request, how do we actually route to the destination avoiding the specified criteria? Since avoidance routing is an on-demand system, the routes must be discovered on the fly. This is done using a *depth-first search*. An avoidance request will transit through the network, searching for a path that matches the given constraints. Once the path has been found, the user can send data along that path. How this all works will now be discussed in detail.

The first step is to construct the avoidance request as discussed in the previous section. After the request is constructed, it is handed to the first router, which will begin the search. It is important to understand that the first router may be the user's own system. So, given a request, what does a router actually do?

We will begin by assuming that the router has not already serviced this request. The router will begin by searching for a known route that satisfies the request. It does this by searching through its advertisements and comparing the security vectors to the avoidance

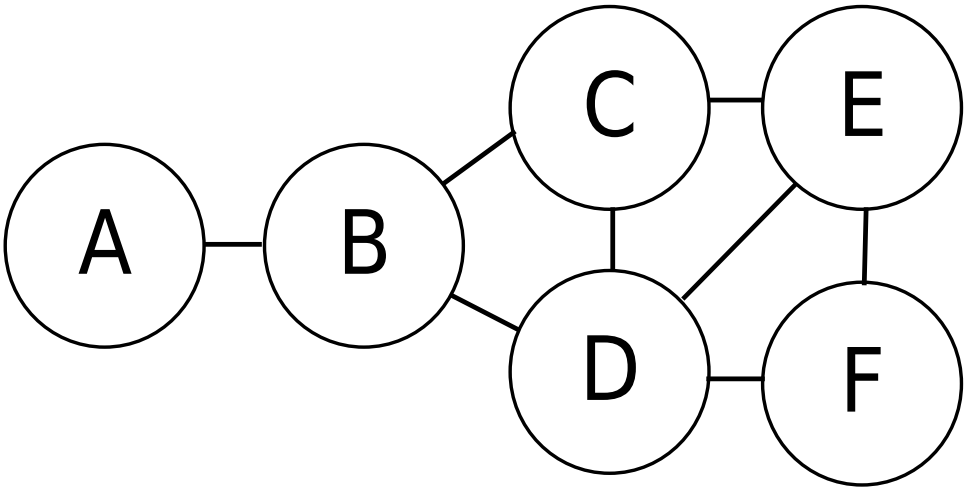

Figure 4.4: Our simple six-node topology

criteria. If there is no collision, that is, the path does not violate the constraints, then this router has done its work. The router simply sends the request to the next router specified in the valid path.

What if the router does not find any valid paths? Does this mean that no valid path exists from this router? The answer to this question is no. Once again, consider our simple topology shown in Figure 4.4. As discussed in Chapter 2, each BGP node only advertises one path for every destination to its neighbors. Figure 4.5 shows A's view of this simple topology. Since node B will only advertise one path to each destination, A only learns of the B – C – E and B – D – F paths. A does not know about the C – D, D – E or E – F links.

Assume A wishes to avoid a property that C has while still reaching node E. If A makes its decision using only the advertisements, it would appear impossible to reach E through node B. However, Figure 4.4 clearly shows that a path exists through B to E avoiding C. Fundamentally, there are unadvertised paths that exist that may satisfy the avoidance criteria.

Not all unadvertised paths are equal however. We identify three classes of BGP paths.

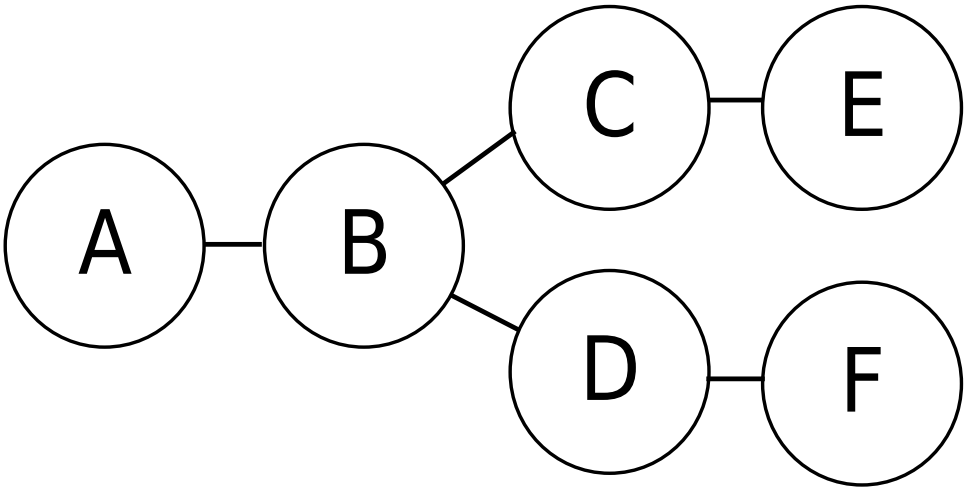

Figure 4.5: Node A's view of the topology

The first class is the **advertised** path. These are the paths that are contained in BGP advertisements and are normally used to route packets. The second class of paths are unadvertised **valid** paths. These are paths that an AS is willing to use to reach the destination, but has not chosen as the path to advertise. Since only one path is advertised, BGP routers must determine which path to advertise. Some paths are chosen simply based on the fact that they are shorter. Other paths may be used in an attempt to load-balance the system. All of these paths could be used to reach the destination, but are currently not used.

The final class of paths are unadvertised **prohibited** paths. These are paths that an AS will never use to reach the destination. These are commonly prohibited due to economic reasons. For example, the no-valley policy would prohibit a provider from ever using a customer as a transit router to another provider. This violates the agreements set up between autonomous systems. Because these paths are prohibited, they are not usable by avoidance routing. For all intents and purposes, these routes do not exist.

Avoidance routing leverages the fact that there are a high number of the second category of routes, unadvertised valid paths, especially in the core. We believe this to be true because BGP has every node advertised to every other node a path to every destination as long as

that does not violate policy. In the completely connected core, where all core nodes are peers, a node should have a valid unadvertised path from each of its peers. In our example, consider the B – D – E path as valid but unadvertised. We would like to use this route, but A has no idea that this route exists. One goal of the depth-first search is to find these routes.

We now know that if none of our current advertisements satisfy the criteria, one of our neighbors may have an unadvertised path that does satisfy the criteria. Our strategy is to give the request to one of our neighbors and let them make the decision. If they have a valid path, the search will be done. If not, they will give the request to one of their neighbors, hoping that neighbor has a valid unadvertised path. Thus, the packet will search the network in a depth-first manner for a valid path.

We only use neighbors that have advertised a path to the destination. If a neighbor has not advertised a path, most likely one of two situations has occured. The first is that we are that neighbor's access to the destination, thus they do not know another route except through us to the destination. The second is that all of their routes are prohibited. That is, we are not allowed to use those routes. In both cases, it does not make sense to try to use these neighbors as they both will fail. It may be possible that there is a third reason that a neighbor has not advertised this path. However, these two described are the most common, and in order to prevent wasted search resources, we do not try these neighbors.

Several questions are still outstanding. How do we determine which neighbor to give the packet to? What happens if that neighbor cannot find a valid path? We determine which neighbor to give the packet to based on a heuristic. If that neighbor cannot find a valid path, it will inform us that it failed, and we will try another neighbor based on the heuristics.

This is actually accomplished by sorting all our neighbors based on a given heuristic. Example heuristics are shortest-path-first or fewest-total-collisions. Heuristics will be discussed in further detail in Section 4.4. Once the neighbors are sorted, we simply give the

Algorithm 1 The Search Algorithm.

```
1: Receive avoidance request
2: Create search table entry
3: for all Advertisements a_x to destination do
4:    if Security vector in a_x ⋔ avoidance criteria then
5:       Forward along interface for a_x
6:       Delete search table entry
7:       Return success
8:    end if
9: end for
10: Heuristically sort advertisements to destination
11: for all Sorted advertisement sa_x do
12:    Forward along interface for sa_x
13:    if Success returned then
14:       Delete search table entry
15:       Return success
16:    end if
17: end for
18: Delete search table entry
19: Return failure
```

request to the first neighbor on the list. If that neighbor returns failure, we move to the next neighbor on the list. If we exhaust the list, then we return failure to the neighbor that gave us the request in the first place. If the request reaches the destination, then the destination will return success. Algorithm 1 shows the basic search mechanism once an avoidance request is received.

The algorithm shown references a search entry, which is shown in Figure 4.6. Some information must be kept during a search. This information includes the source of the

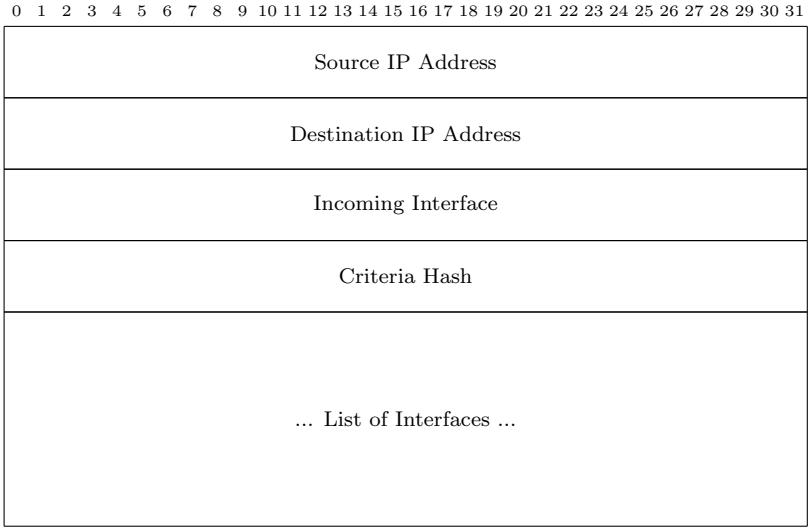

Figure 4.6: The search table entry

avoidance request, the destination, the avoidance criteria, and an ordered list of interfaces yet to be tried. As the search is conducted, each tested interface is removed from the list. If the list becomes empty, then no interface succeeded and failure is indicated. Finally, the interface on which the search arrived is also stored.

This information is stored for three reasons. The principle reason is to store the sorted list of interfaces for the search. The source, destination and avoidance criteria also work as a three-tuple to identify the request. This way, any request can be tied to a given search entry. This also prevents us from creating duplicate search entries if a source attempts the same request multiple times. It may also be useful if future avoidance routing solutions use breadth-first mechanisms. Since the avoidance criteria can be a variable length, we store a hash of the criteria instead. This allows us to save space while still allowing us to identify requests. The final reason to store this information is to prevent search loops. If the request arrives on an interface other than the one original interface, we can infer a loop occurred and return failure to the looping neighbor. This behavior would have to be modified to support breadth-first mechanisms.

The algorithm also shows places where a node returns success or failure. These are

explicit messages that a node will send to the neighbor that sent the request. If a node cannot continue the search, it will indicate failure to the previous hop with a failure message. If the destination receives the request, it will indicate success which will propagate back along the path to the host. These messages are sent explicitly. When discussing the avoidance requests, all forms had a message type field. For an avoidance request, this field is set to "request," and the message contains the avoidance criteria. For success and failure messages, this field is set to either "success" or "failure." Thus, the headers shown in Section 4.1 are general avoidance routing headers.

One important caveat is that our system does not consider path-dependent properties. A path-dependent property is any property where the routing decision is dependent on the path used to reach the current router. For example, delay is a path-dependent property. Path-dependent properties greatly increase the complexity of a search by reducing the pruning of already visited nodes. If we arrive at node C from node D and the search fails, this does not mean that if we arrive at node C from node B, the search will fail.

We could possibly handle path-dependent properties by using partial searches. Each time a node receives an avoidance request, it conducts a search regardless of whether it can currently satisfy the path-dependent properties. In this way, the search data is available and usable for pruning if the search reaches this node from another path. Another possibility is to use a limited crank-back approach as discussed by Shin et al. [SCS01], which limits the number of redundant searches based on several factors, such as the number of searches already conducted. However, because of the complexity of the problem and our desire to have a high-speed system, we do not consider path-dependent properties.

4.2.1 Caching

It is clear that conducting a search for every packet is impractical, and with larger avoidance requests it is impossible. Further, every packet in a particular data flow will want to take the same avoidance path. Searching for a path for each packet is inefficient, since the

first packet will have already discovered a viable avoidance path. Finally, since avoidance requests may require an entire packet, routers must be able to recognize packets that wish to use avoidance routing without needing to know their avoidance criteria every time. To solve this issue, we employ caching and MPLS-style techniques. The goal is to ensure that once an avoidance path has been created, all subsequent packets can be forwarded via the cache without specifying avoidance criteria each time. In this way, the search is only conducted once.

We refer to our caching mechanism as the *forwarding cache*. This cache is organized by an index number and contains only three pieces of information: a destination, a forwarding interface, and the index number for the next router. It is a fixed size with fixed-sized entries to allow for quick forwarding look-ups. When a route is successfully discovered, a cache entry is created and stored in the next available cache slot. Since the cache is a fixed size, it is possible that no slots are available. In this case, the least-recently-used (LRU) entry will be evicted, and that slot will be used for this path.

The index number of the cache slot is communicated to the previous router in the success message for the avoidance path. That router, in turn, will create a cache entry, storing the index number of the next router. In this way, the source will receive a success message containing the index number of the first router along the path.

We employ a MPLS-like mechanism to actually forward the packets. When the source uses the service, it places the index number of the first hop in the IP header as an IP option. The first router receives the packet, and looks up the forwarding interface based on the index number. The router changes the current index number in the packet to that of the next router and forwards it along the specified interface. In this way, the avoidance path can be used without every packet containing the explicit avoidance criteria. Since the forwarding cache has small fixed-sized entries, we believe the cache can eventually sit directly on the router using high-speed memory. Therefore, the forwarding cache allows quick high-speed forwarding of packets.

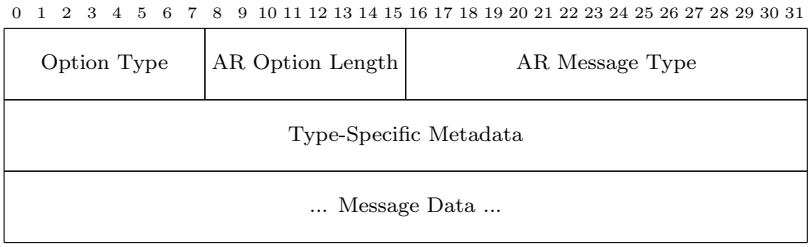

Figure 4.7: The header form

One important optimization in the forwarding cache is that reverse entries are created when an avoidance request arrives. This cache entry is identical to a normal cache entry except it is set up in the reverse direction (to the source), unless routing policy does not permit the router to send packets in that direction. Logically, the packet could not have reached this router from the source unless an avoidance path exists to this router. The previous router will place its cache index number in the avoidance request before forwarding it to the next router.

Both success messages and avoidance requests now require a field to store a cache ID. Recall the avoidance routing message formats discussed earlier. Figure 4.7 shows the header form message again for reference. In these messages there is a field for "type-specific metadata." This field is designed to hold specific data for each message type. For an avoidance request, this will be the cache ID of the router forwarding the request. In a success message, this will be the cache ID of the router indicating success. In this way, each router will learn the cache IDs of its neighbors as an avoidance path is built.

Success or failure messages also have message data. This data is a hash of the avoidance criteria given in the avoidance request. This allows routers to quickly find the search entry that corresponds to the given avoidance request. Further, it makes it more difficult for an adversary to try to spoof success and failure messages. This security concern will be discussed in more detail in Section 4.6.

Figure 4.7 also shows the format for an avoidance routed packet. In this case, the

message type would be "forward" and the type specific meta data would be the cache ID of this router. This is the cache ID that would be used for the forwarding look-up and would be replaced by this router.

4.2.2 Path Invalidation

The description of the forwarding mechanism makes it clear that forwarding will work if everything else works correctly. But what if the cache entry referenced in the packet does not correspond to an actual cache entry? What if the cache entry is for another avoidance path? What if the interface specified in the cache entry is invalid due to a routing failure or policy change?

These problems are overcome with a second level of caching, referred to as the *avoidance cache*. This cache is organized by destination and stores the following information: destination, forwarding cache index, avoidance criteria, next router index number, and forwarding interface. While it may appear that the avoidance cache makes the forwarding cache redundant, there are two important distinctions. First, avoidance cache entries are larger. If we only used an avoidance cache, we would store less paths per byte of high-speed memory. The second distinction is more critical. Since avoidance criteria are of variable length, it is impossible to know a priori the size of cache entries. This makes memory layout and management a more difficult problem. For these reasons, we deploy the forwarding cache which uses small fixed-sized entries, making memory layout and access far easier.

It may also appear that the avoidance cache is unnecessary given the forwarding cache. However, the avoidance cache serves several purposes. Recall that the forwarding cache stores the destination. The first purpose is to handle a packet whose destination does not correspond to the forwarding cache entry based on index number. This is validated while the cache ID is being updated. The entry may disagree with the packet's destination if the forwarding cache entry was evicted. In this case, the avoidance cache is checked, the packet can be quickly forwarded, and the forwarding cache can be updated.

The next purpose, and possibly the most important, is to store the avoidance criteria. As stated above, the forwarding cache uses an index system so that individual packets do not have to contain the avoidance criteria. However, this criteria must be maintained in case there is a routing change or other path invalidation. When a routing change occurs along an avoidance path, that path is no longer valid. The last still-connected router will be the first to detect the routing change, and thus the invalid path. At this point, this router will conduct a new search to the destination using the cached avoidance criteria.

This is also important for resurrecting avoidance criteria from a previous router. If a packet using avoidance routing arrives, and it does not have either forwarding cache entry or an avoidance cache entry, the router can request that the previous router conduct a new search. Not only does this router have a valid avoidance path, it also has the avoidance criteria. In this way, it simply generates a new avoidance request and sends it along its cache's interface to the router that does not have any cache entries. This router now has the avoidance criteria and can conduct a new search.

Algorithm 2 The Search Algorithm with Caching

```
1: Receive avoidance request ar
2: for all Avoidance cache entry ac_x with destination d in ar do
3:    if Criteria c in ac_x = criteria in ar then
4:       Forward along interface specified in ac_x
5:       if Failure Returned then
6:          Delete ac_x
7:          Goto 14
8:       else
9:          Create Forwarding Cache Entry
10:         Return success
11:      end if
12:   end if
13: end for
14: Create reverse cache entry
15: Create search table entry
16: for all Advertisements a_x to destination do
17:   if Security vector in a_x ∌ avoidance criteria then
18:      Forward along interface for a_x
19:      Delete search table entry
20:      Create cache entries
21:      Return success
22:   end if
23: end for
24: Heuristically sort advertisements to destination
```

25: **for all** Sorted advertisement sa_x **do**
26: Forward along interface for sa_x
27: **if** Success returned **then**
28: Delete search table entry
29: Create cache entries
30: Return **success**
31: **end if**
32: **end for**
33: Delete search table entry
34: Create failed avoidance cache entry
35: Return **failure**

The final purpose of the avoidance cache is for search optimization. When an avoidance request arrives, the router checks to see if a cache entry to the destination with the specified avoidance criteria already exists. This includes the case where the new criteria is the same as the stored criteria, but also if the new criteria is a subset of the old criteria. If so, it can use the existing forwarding interface to forward the message along without searching. This may result in better search performance at the penalty of using a suboptimal path (especially in the subset case). This can also be used to store **failure**. In this case, a recent search for a specific avoidance path to a destination failed. This allows us to prevent search loops and prune branches that have already been used by a previous search. Algorithm 2 shows how caches modify the search mechanism, while Algorithm 3 shows how caches work for forwarding.

Unfortunately, because the avoidance cache has variable length entries, it is impossible to say how much space the cache will take. We expect that the largest part of the cache entry will be the variable length avoidance criteria. Since avoidance criteria may be the same for many paths, it may be possible to save only a master copy of that criteria and have all other cache entries with the same criteria point to the master copy. In this way,

Algorithm 3 Using Caching to Forward a Packet.

1: Receive packet using avoidance routing
2: Get cache index from packet
3: Get forwarding cache entry fc_i
4: **if** fc_i exists **AND**
5: fc_i destination = packet destination **then**
6: Forward along specified interface
7: Return
8: **end if**
9: Get avoidance cache entry ac_i
10: **if** ac_i exists **then**
11: Forward along specified interface
12: Create new forwarding cache entry
13: Inform previous hop of cache change
14: Return
15: **end if**
16: Request previous hop to generate new avoidance request
17: **run** Algorithm 2

much space could be saved.

Finally, it's possible that security properties for a node change. While we expect this to be rather unlikely, it is still possible. If a node detects that its neighbor has changed security properties, it must first check to see if any of its paths are invalid. It will look through its avoidance cache for any paths using that interface. For paths that do use the interface, it will have to check to see if the new or modified property violates the criteria. If so, it will have to begin a new search as if the link had failed.

Due to the possibility of routing failures, we have included several messages that are not required to be used, but may be helpful. These message types indicate to previous routers

that a new search is being conducted. These messages can propagate all the way back to the source, who may wish to throttle data transfers until the path has been reconstructed. These messages are not required for avoidance routing to work, as routers can infer that a new search is being conducted if avoidance requests or success/failure messages arrive.

4.3 Policy

One of the fundamental reasons the Internet uses BGP is its ability to express policy. Policy describes what paths to advertise to your neighbors. As we discussed in Section 4.2, there are prohibited, unadvertised and advertised paths. An example of a prohibited path is one that arrives from a provider and attempts to use a provider path. The no-valley policy describes on of the reasons that paths are deemed available or prohibited.

Figure 4.8 represents the hierarchical Internet graph with examples of the no-valley policy. Here, Nodes S, O and D represent customers to nodes I-1 and I-2, which represent the intermediate nodes. I-1 and I-2 are both customers of the core node. The path going from S up to the Core and back to D is a valid no-valley path. The path from S to I-1 to O to I-2 to D is not a valid no-valley path as O receives data from its provider I-1 and is expected to forward the data to its provider I-2.

One of the fundamental principles of avoidance routing is not to use path-dependent properties. This is because the search space would explode if a node has to conduct a new search based on the path taken to reach that node. In effect, policy is a path-dependent variable. If the search arrives from a customer, we can use any neighbor. Since the search is going "up," we can use providers, peers or customers. However, if the search arrives from a provider or a peer, we can only use customers. This is because the search is going "down." Thus, if the search first arrives from a provider, fails, and then later arrives from a customer, we must actually search again as new neighbors are now valid and available.

Luckily, this isn't quite as bad as a fully path-dependent variable. Assume that we wish

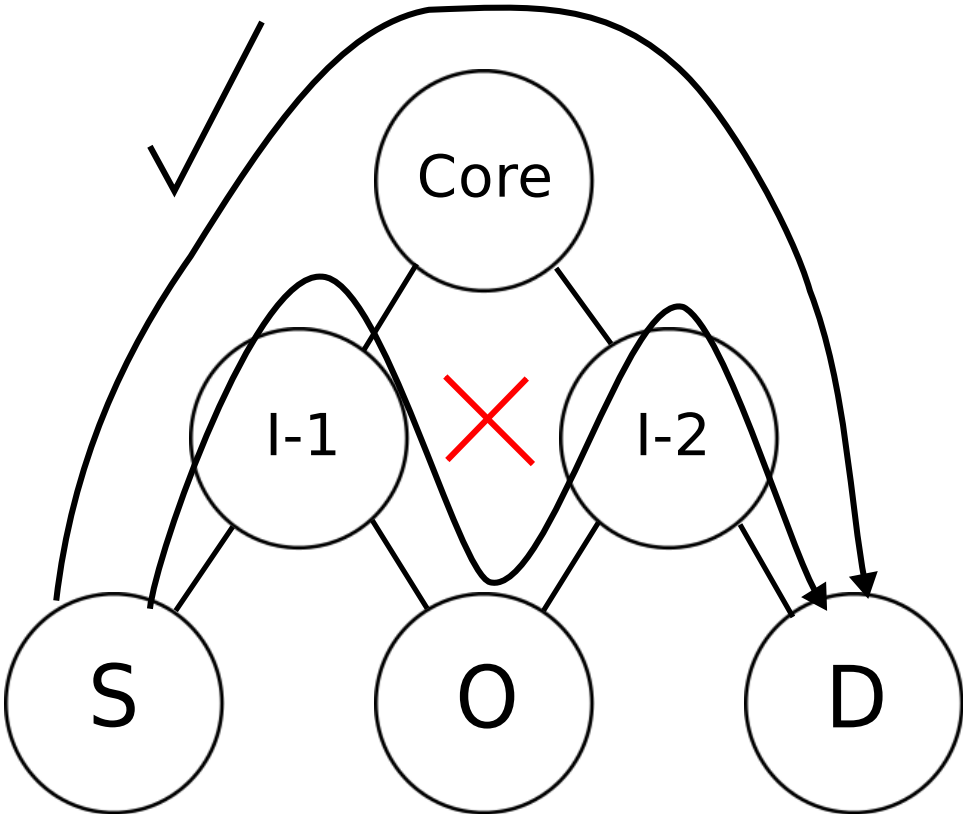

Figure 4.8: The no-valley policy

to keep delay under 1 second. Delay is a path-dependent variable. A request arrives at a node with a delay of 900 ms. It tries all its neighbors, but they all fail, so it reports failure back. However, the request arrives again with a delay of 200 ms. Now, all neighbors need to be tried again. Fortunately, this is not the case for policy-based avoidance routing.

In avoidance routing, if a request arrives from a provider or peer, we can only try customer nodes. If the request later arrives from a customer, we can now try all nodes. However, we already tried our customers, and they failed. If we try them again, they'll still fail since they can only retry their customers (since we are their provider). Thus, in this situation, we only need to try peers and providers, i.e., the nodes we haven't tried. This means that, at worst, a node needs to run the search twice, but each time it searches, it will try different nodes than it tried before.

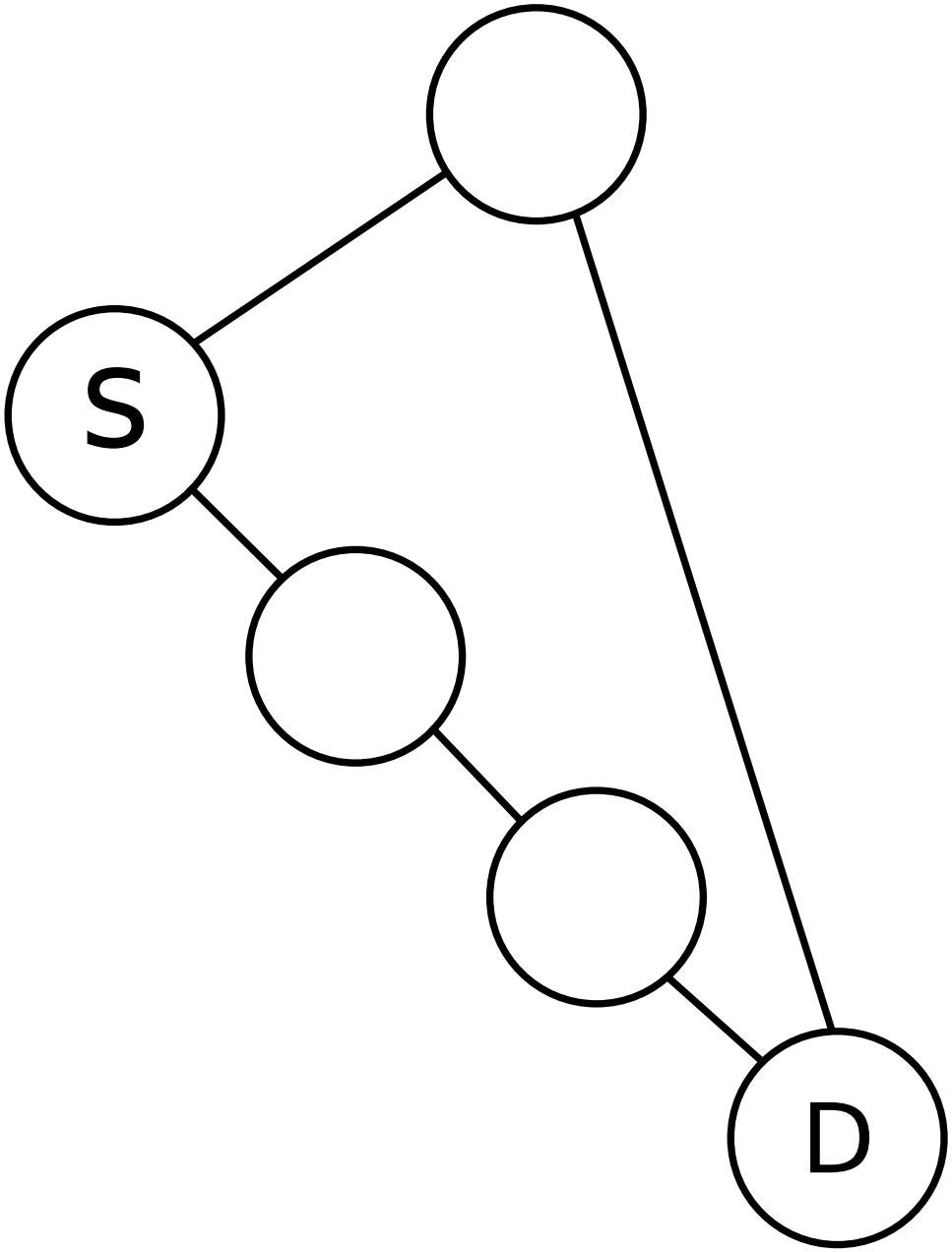

Figure 4.9: Favor customers

Standard BGP policy also has another component, “favor customers.’. The topology shown in Figure 4.9 will help demonstrate this policy. Here, “up” links are provider links, and “down” links are customer links. The favor customers policy says that you always prefer to go downhill to a destination rather than uphill. In this example, S has two paths to D, a

longer path through its customer or a shorter one through its provider. The favor customer policy says that S will always choose its customer, even though the path is longer. This is because that customers pay providers to move packets. Thus, if you use your customers, they pay you money. If use your providers, you pay them money. Therefore, it is always to your advantage to go downhill through customers as long as you won't go uphill later.

This policy makes no difference to the complexity of the search. Favoring customers does not mean you have to search multiple times from one node. However, it does affect which neighbors to try first. If you use the favor customer policy during the search, customers should always be tried before providers, regardless of the heuristic. We implemented this mechanism into avoidance routing and evaluated it effect on the length of the search. This is discussed in Chapter 5.

4.4 Heuristics

A depth-first search that does not incorporate any additional information may experience significant additional overhead as it blindly traverses the network. To help avoidance routing find paths quickly, we annotate the topology as discussed in Chapter 3. This information can clearly be used to find valid paths immediately and use them without conducting the search. How to use this information to find better paths when there is not a clear valid path is a question we tried to answer in this research.

Many search algorithms employ heuristics to help prune the search space. These heuristics leverage information about the search space to do this pruning. For example, in a mini-max algorithm, alpha-beta pruning is employed to remove sub-trees that are quickly determined to be suboptimal. If a sub-tree only has values greater than all other values in another sub-tree, the first tree is not useful during the minimization phase and can be pruned.

Avoidance routing attempts to employ heuristics that we feel make sense for our search

space and the information that has been provided. These heuristics include shortest path first, furthest collision, nearest collision and least total collisions. Remember that a collision occurs when the avoidance criteria intersects the security properties of a router. For example, a collision will happen if I want to avoid France and a router on the path is in France. Any path via an invalid neighbor is immediately excluded. In this section we will discuss our different heuristics. We will discuss the reasoning behind the heuristic, as well as cases where the heuristic may succeed or fail. The heuristics discussed here are not an exhaustive list. There may be many more heuristics that could be used, and we welcome further research in this area.

4.4.1 Neighbor Only

Before we discuss the other heuristics, let us remind you about the neighbor-only variant of avoidance routing. As discussed in Section 3.5.1, ASes may not wish to disclose all the information that we would like to have. Further, ASes may want to limit who has access to this information. Thus, we created the neighbor-only variant of avoidance routing. This variant requires that neighbors are the only nodes that need to know one's security properties. This is the minimum amount of disclosure required for avoidance routing to work.

Neighbors are required to know this information so that they can differentiate between valid and invalid nodes. Without this information, the only way to make this differentiation would be at the node itself, which would violate the goal of avoidance routing. This variant is roughly equivalent to a blind, or non-heuristic based, depth-first search. It is impossible for any paths to be prioritized or a priori pruned, since a node only has information about its neighbors.

Neighbor-only avoidance routing does have access to all the information available in standard BGP. Since path lengths are known in BGP, this variant can select paths via path length heuristics. In this way, neighbor-only avoidance routing could use the shortest

path first heuristic.

4.4.2 Shortest Path First

The shortest path first heuristic is the simplest and most standard heuristic. Except for economic-based policies, the Internet generally operates by selecting the shortest path. The theory is that the faster data reaches its destination, the less load the entire network will experience and the more overall traffic can be served. By selecting shortest paths, it's likely that these will be the fastest paths.

But, this is not necessarily the case. While the shortest path may generally be the fastest path, the shortest path may transit a bottleneck link. These are links with low bandwidth or that are in high demand. In this case, a longer, less utilized, higher bandwidth path may actually be the fastest path. Unfortunately, bandwidth and utilization information are not currently shared in BGP. To overcome this limitation, ASes use a technique called AS path prepending which artificially increases the length of a path by adding itself to the path multiple times. Thus, if a link is over-utilized, an AS can make its paths arbitrarily long until another path is the shortest path.

Avoidance routing uses the same basic assumption but for slightly different reasons. We assume that the faster one reaches the destination, the less likely a collision can occur. Assume that all security properties are uniformly distributed. The odds of a collision are $N * p_x$ where N is the number of nodes on a path and p_x is the probability that a node has property x. Thus, smaller values of N are less likely to experience a collision.

Unfortunately, the fundamental assumption about uniform distribution is false. Some properties may experience uniform property distribution, like AS number or hardware manufacturer. Other properties, like geopolitical location, will not be uniformly distributed. If one wishes to avoid the United States but the shortest path intersects the United States, its likely that any nearby node will also be in the United States. In this case, this heuristic might actually make the situation worse.

4.4.3 Furthest Collision

The furthest collision heuristic leverages a very simple principle — the farther a collision is on a path, the more likely an alternate valid path can be found. Basically, if I take a route to the destination where something I distrust is far away, I may find a good path long before I reach the untrustworthy node. This heuristic has one major negative flaw; favoring longer paths over shorter ones. A path of length 6 with a collision on the last node will appear better than a path of length 5 with a collision on the last node. This is the heuristic's advantage and disadvantage. It may be very difficult for this heuristic to get "trapped" and waste a lot of resources, but it will also tend to find suboptimal paths.

When would this heuristic be useful? We assume this heuristic is useful when attempting to avoid a large contiguous region, such as the United States. Heading straight for a large mountain range may make it difficult to find a clear path around it. Further, you might get stuck in a valley and have to backtrack. However, heading away from the mountain range may help you find an easy path to the destination, even if it is a longer one.

Clearly, the heuristic will be bad in the opposite case. If you're trying to avoid something uniformly distributed, heading away from a collision will probably just drive you closer to another one with little overall benefit. Our results, shown in Chapter 5, discuss how this heuristic works in practice.

4.4.4 Nearest Collision

Nearest collision is the opposite of furthest collision. Rather than trying to find a path furthest from a collision, this heuristic drives right for a collision. Nearest collision derives from the reverse logic of furthest collision. If a colliding property is small, like router manufacturer, it may be very easy to find a path around the collision. Thus, we expect this heuristic to operate exactly opposite to furthest collision. If we are trying to avoid something uniformly distributed, heading right towards a collision will most likely find a

path around the collision. However, heading towards a large contiguous region may result in the search getting "trapped" and having to backtrack, wasting time and resources. How these two heuristics relate and actually play out together will be discussed in Chapter 5.

4.4.5 Least Collisions

Least collisions operates on a different principle from the previous two heuristics. While those heuristics were considering the distribution of properties in the network, this heuristic considers the distribution of properties on a path. The hypothesis here is that the less collisions that occur on a path, the more likely that an alternate avoiding path can be found.

Consider the case of the United States. If the US lies between you and your destination, it is likely that it will appear multiple times on the path. Thus, a path that has less collisions may be around the edge of the US rather than transiting right through it. Further, if the property you care about is uniformly distributed, how many times it appears on the path should not affect the likelihood of finding an avoiding path. Thus, using this heuristic should not be negative for uniform distributions and may benefit for contiguous distributions.

This property may also naturally provide a shortest path first mechanism. Shorter paths are less likely to have collisions. A path with the least overall collisions will likely be short. In this way, this heuristic will provide us with shortest paths.

4.4.6 Prioritized Heuristics

So far, we have discussed four heuristics independently. There is nothing about these heuristics that says we could not use them in consort. Since advertisements are sorted by heuristics, we could have primary, secondary and tertiary heuristics. Our evaluation will look at the heuristics individually, as well as combining some based on the individual results. Some heuristics do not make any sense to combine. The furthest and nearest

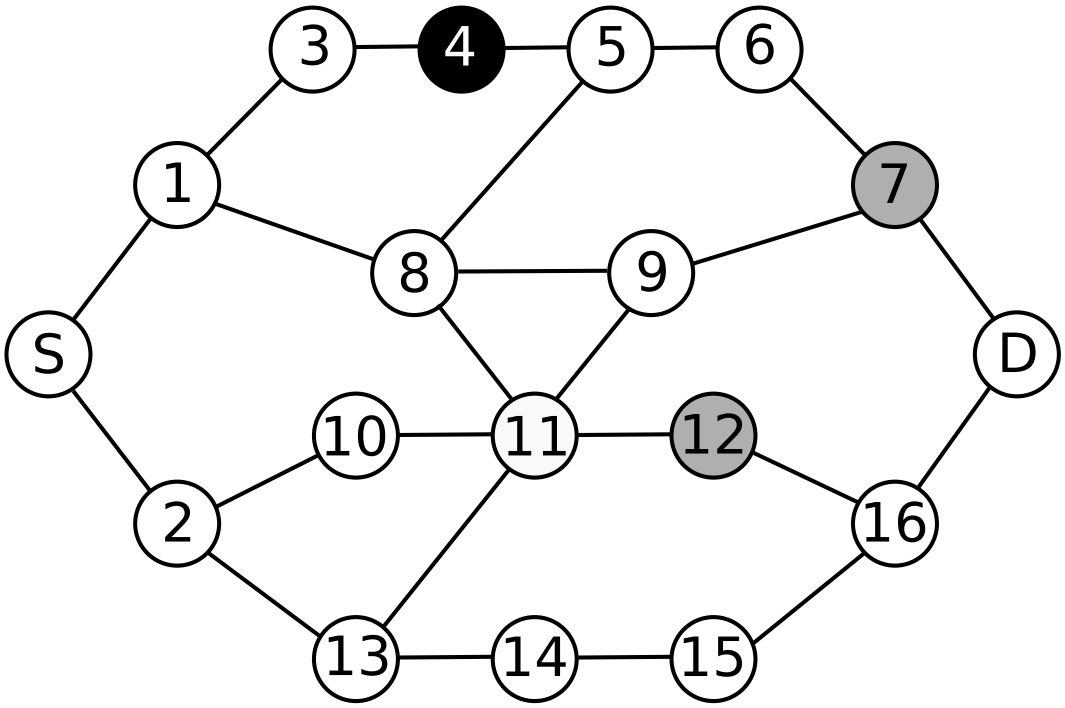

Figure 4.10: An example topology for avoidance routing

collision heuristics are opposites; sorting by one followed by the other will not produce any new results; it will just be a waste of time and effort.

4.5 Example of Avoidance Routing

Now that the main concepts and mechanisms of avoidance routing have been introduced, we feel it would be useful to walk through an example of how all those concepts and mechanisms work in practice. For this example, we will be referring to the topology shown in Figure 4.10. In this example, source node S wants to send data to destination node D while avoiding nodes with the gray property. In this case, nodes seven and twelve are the only nodes with the gray property. The black node is a node that is not participating in avoidance routing or has not specified any information regarding the "gray" property.

In this topology, all nodes on the left side of a link are a providers for the node on the

right side of the link. That is, node S is a provider for node 1, which in turn is a provider for nodes 3 and 8. We are assuming a "no-valley" policy as described in Section 4.3. In "no-valley", once you are going "downhill," that is from provider to customer, you cannot go back "up" to a provider. For this example, it means that if we arrive at node 11 from node 10, we cannot go to nodes 8 or 13 since they are also node 11's providers. We can still use 9 and 12 as they are customers.

The search begins with the user at node S sending an avoidance request to node S. This request will specify D as the destination with the avoidance criteria of gray. Node S will begin by checking its cache for any information on paths it may already have. In this example, there are no nodes with cached information for this request. Next, node S will determine if it already knows about a valid advertised path to D. The topology would suggest that S has a fully valid path through 2, the {*2 - 13 - 14 - 15 - 16 - D*} path. In this example, this is not the case. Neither the path advertised by node 1 nor the path advertised by 2 are valid. Figure 4.11 shows the paths advertised **to** nodes 1 and 2 by their neighbors (3, 8, 10 and 13). As one can see, all the paths to D that 1 and 2 are aware of transit a gray node. No matter which path 1 and 2 choose to advertise to S, it will transit a gray node.

For this example, we will assume that we are using the shortest path first heuristic (SPF) to determine which neighbor to try first. Node 1 advertised to S the {*1 - 8 - 9 - 7 - D* } path, which is shorter than either path that 2 could have advertised. Thus, S will choose to try node 1 first, forwarding the avoidance request to it. When 1 receives the avoidance request, it too will look to see if it already has a completely valid path. As Figure 4.11 shows, node 1 does not know a valid path to D. Using SPF, 1 will try node 8 next.

Figure 4.12 shows the routes to node D that node 8 is aware of. Clearly, 8 knows three paths to reach D, through nodes 5, 9 and 11. None of these paths are valid however. Using SPF, 8 will try 9 first. Node 9 has only one downstream neighbor it can use; however,

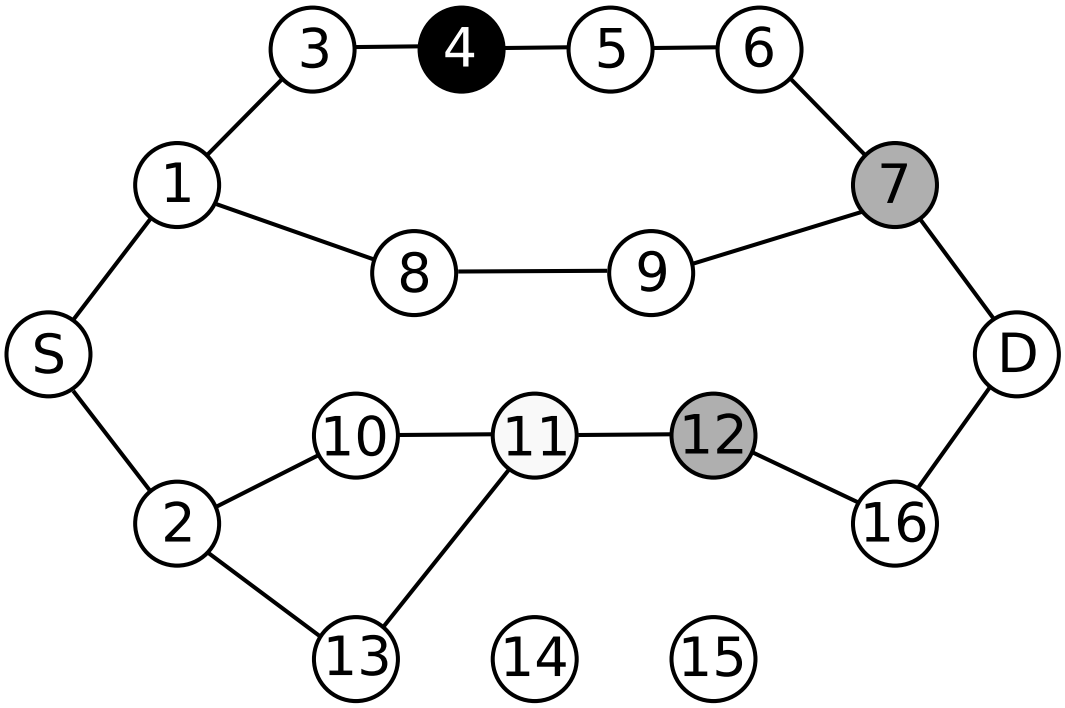

Figure 4.11: The routes to node D that nodes 1 and 2 are aware of

this neighbor (7) violates the criteria. Node 9 marks failure in its avoidance cache and return failure to node 8. When 8 receives the failure message, it will try its next neighbor according to SPF. Paths through both 5 and 11 are equal length. At this point, node 8 may use a secondary heuristic like furthest collision. Since node 7 is further than node 12, it would try node 5 first.

Node 5 has only one option towards D, and forwards onto node 6. Node 6, like 9, can only reach D through 7, and must send failure to 5. Having exhausted its only option, node 5 returns failure to 8. Nodes 5 and 6 will have also made negative cache entries in their avoidance cache. When the failure message reaches node 8, it is left with one final option to try, node 11. Node 11 can only try one neighbor, node 9, as 12 is an invalid node.

Node 9 will receive the avoidance request, check its cache and find the negative cache entry. Thus 9 will not try to search anything and immediately returns failure to 11. Upon receiving failure, 11 will conclude failure and propagate this to 8, which has also concluded

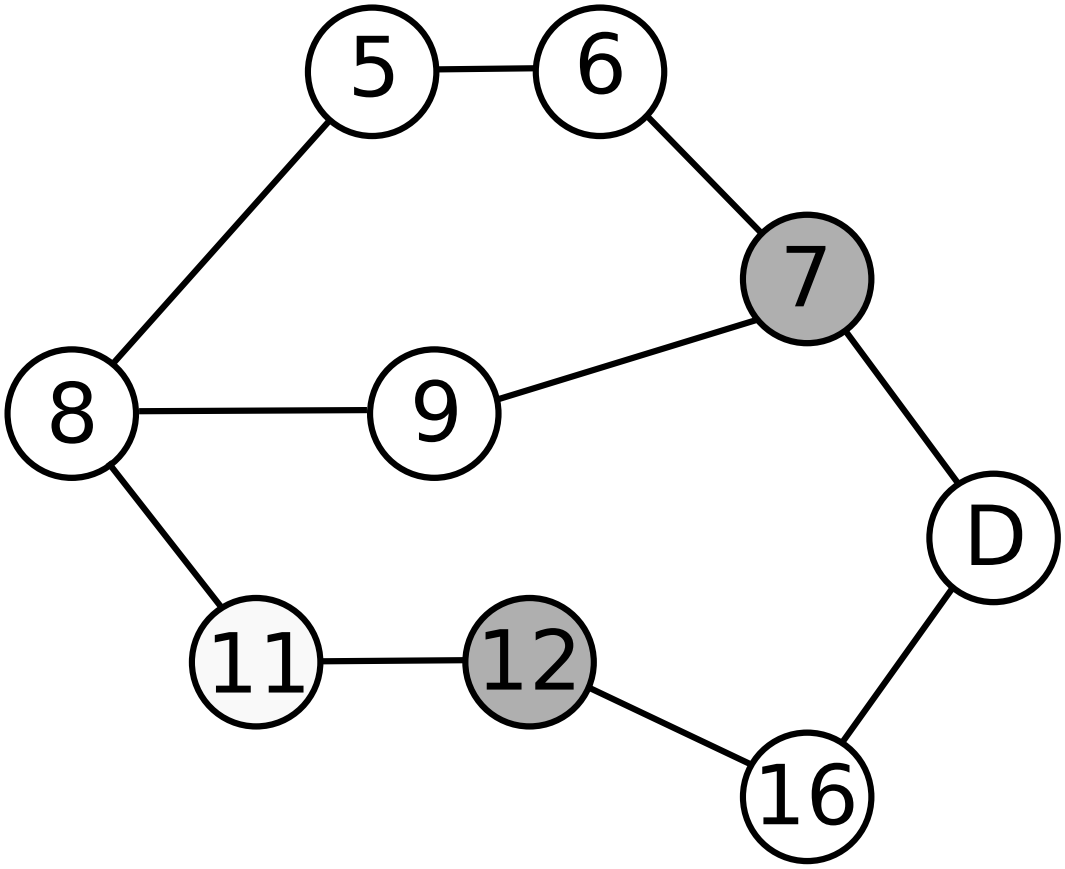

Figure 4.12: The routes to node D that node 8 is aware of

failure. Node 1 will finally receive the failure message, and be left with the only option of trying 3. Node 3, as 6 and 9 before it, will receive the request, determine its only neighbor is invalid and will return failure. It is important to note that node 4 is invalid not because it has the gray property, but because it is not participating in avoidance routing. Node 4 may actually not have the gray property, but since we cannot determine this, we cannot use this node.

With the failure at 3, node 1 will have exhausted all of its options and return failure to S. All nodes touched so far (1, 3, 5, 6, 8, 9, 11) will have marked failure in their avoidance caches. Node S will now try its final option, node 2. Node 2 is also not aware of a direct path to D. Neither the SPF heuristic nor the furthest collision heuristic will break the tie

between nodes 10 and 13. At this point node 2 will pick either neighbor at random. In our implementation, it will choose whichever node sent an advertisement to it first, since this ad will appear higher in the RIB if all else is equal.

We assume node 10 is chosen. Node 10 has one option, node 11. However, 11 has already been tried, exhausted its only neighbor 9 and has a negative cache entry. Thus, 11 immediately informs 10 of the search failure. Node 10, in turn, informs 2, which is left with node 13. When the request arrives at 13, we finally have a node that knows a completely valid path — through node 14. Thus, 13 immediately forwards through node 14, who in turn forwards to 15, 16, and finally to D.

Receiving the avoidance request, D determines that the request is destined for it, and returns the success message to 16. This message will propagate along the discovered path, {*16 - 15 - 14 - 13 - 2*} and finally back to S. As each node along the path receives the success message, it will clear its search information and place the previous hop in its forwarding cache. Further, the reverse path forwarding entries will have been set up as the avoidance request propagated the network.

Finally, S can start sending data to D. Node S will add the cache ID of node 2 into the avoidance header option and send the packet to 2. Node 2 will look up this in its cache based on the ID in the packet, and swap the ID with that of node 13 and forward to 13. This look-up and swap will continue until the packet reaches D. This mechanism will continue for all packets from S to D using this avoidance path.

Before concluding this example, let us change our assumptions a bit. We had stated that valid neighbors were only those to the right of the current node. This was due to the "left to right" provider-to-customer relationship. What if we removed this limit? Thus, when node 9 received the request from node 8, it could have tried node 11. Recall that the request went from, 8 to 9, to 11. If 9 tries 8, it could lead to a routing loop. This is prevented by storing the source interface and the current testing interface. If a node is in the middle of a search and the request arrives from an interface other than the source

interface (due to a retransmit), it must be a routing loop. What if the message loops around and comes back at 8 through the source interface — through node 1 in this case? This is also impossible. When the message first creates the loop (arrives at a searching node), it must arrive from a different interface than the source interface. Thus, this mechanism prevents the potential for routing loops.

4.6 Security

Whenever someone proposes a technology connected to the Internet, they should consider the security implications of that technology. This is especially true of technologies that are supposed to improve security. Adding these technologies should not open up new avenues of attack for adversaries. An example of a technology with a negative security externality would be one that requires a router to validate the cryptography of every packet. This would be designed to prevent spoofing, but the added work could be used to run a denial-of-service attack on the routers themselves.

We must consider the same implications for avoidance routing. As far as we can tell, avoidance routing cannot provide a new denial-of-service vector against end-clients. Avoidance routing does not cause data amplification, nor does it require work on the end-host's machine. From a receiver's point of view, a packet arriving via avoidance routing is no different from a packet arriving normally. Further, an avoidance request arriving at an end-host is simply responded to by a success message. This would be equivalent to a SYN-ACK without TCP state, or an ICMP echo reply.

Avoidance routing *does* require routers to perform some heavyweight operations. How this could be used as an attack vector against other users or the routers themselves is discussed in the next section. How we prevent this from being a problem will also be discussed.

4.6.1 Using the Search for Denial-of-Service

The example we gave of a negative security externality involved packets forcing routers to use all their resources. Clearly, avoidance routing could be used in the same way. The obvious mechanism is for a malicious avoidance routing user to make hundreds or thousands of avoidance requests. Each request would begin a search, using memory and CPU on any router the search traversed. If an adversary was able to produce a suitably accurate topology, they may be able to make avoidance requests that are guaranteed to fail, maximizing the nodes affected by the search.

An adversary could also make valid requests. Instead of trying to affect the most nodes, these requests would use up cache space. As the adversary built up more valid paths, any normal user's paths would be evicted from the cache using our LRU mechanism. The adversary could consistently send data down these paths to ensure there liveness, as well as forcing reconstruction of any path that got evicted. A suitably knowledgeable adversary could construct avoidance requests that can only be satisfied by transiting the same router. This would allow a distributed adversary to amplify the DoS effect on users of this router and on this router itself.

A final mechanism would leverage what happens when the destination receives an avoidance request. If a destination has security criteria they care about, the criteria are unioned with those of the sender and a new path is built. An adversary who is aware of destinations with their own criteria could selectively choose these destinations. This would increase the effect of router denial by making one search result in two. A very clever adversary could choose criteria that is satisfiable for his request, but is not satisfiable when unioned with the destinations. This would create multiple "maximized" searches and wasted cache space at the same time.

The ability of an adversary to use avoidance routing as a DoS mechanism on the network is clear. Without the means to prevent this, avoidance routing is not practical. The high

goals of avoidance routing would be completely minimized by the potential for destruction. Thus, this destructive capability must be addressed. We solve the DoS problem via rate limiting.

4.6.2 Rate Limiting

To justify the expenditure required to deploy avoidance routing, we expect that service providers will charge their users for service. In turn, tier 1 and 2 providers will charge their customers to use the service. Finally, peers will negotiate exchange rates for using the service the same way they negotiate rates to move packets.

To prevent denial-of-service, each agreement would specify how many avoidance requests a customer could make. These limits would propagate throughout the network. A tier 1 provider would limit its tier 2 customers, and so on. Thus, an individual user could not send enough requests to DoS the system. The rate limits would also prevent an adversary from performing DoS attacks on routers. The router would limit the number of outstanding requests they would honor in any one time period to prevent over-utilization. The combined effect would require an attacker to have a lot of resources in order to cause a DoS effect.

When an avoidance request arrives from a node that has already reached its limit, a limit-reached message is sent back. This message uses the format similar to the success or failure messages and indicates that the limit is reached. A router may choose to always send limit-reached messages rather than honor avoidance requests. This would allow a router to pretend it runs avoidance routing even if it does not. Detecting a router who is violating this constraint is not difficult, as the limits are negotiated. A sender can keep track of the number of successful and limited messages it sends and determine if a router is violating their agreements.

An ISP should only sell enough service to meet the rate limits of their providers. That is, they shouldn't oversell the service so that they can easily overwhelm their providers. Clearly, not all users will use the service at the same time, so some overselling may be

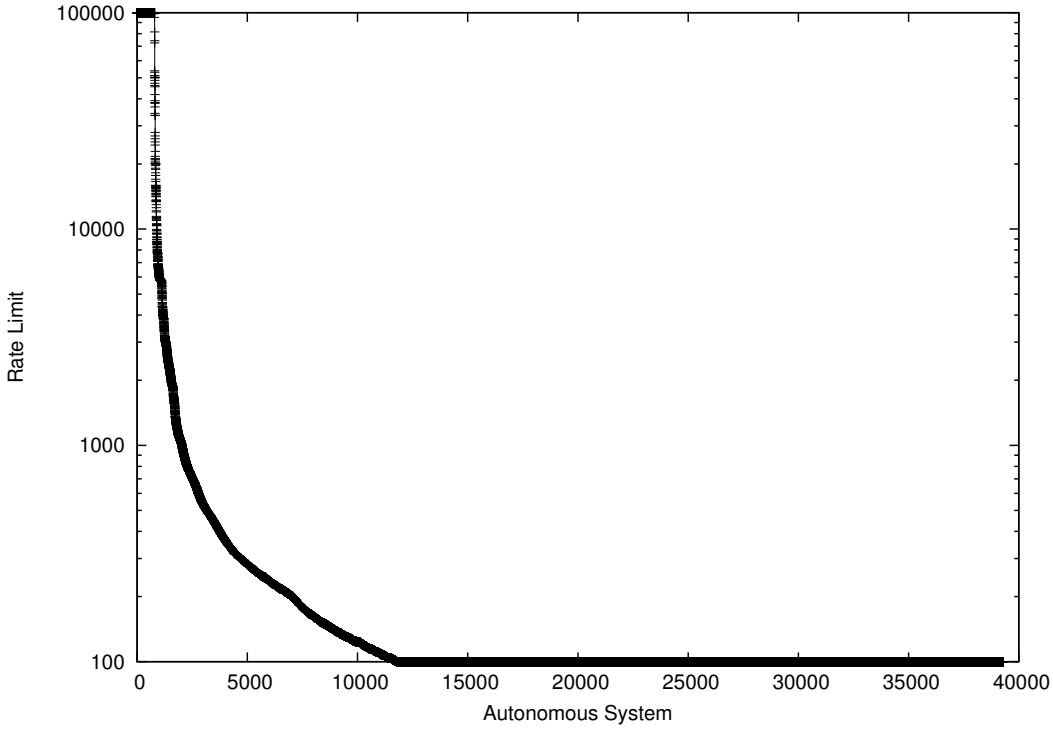

Figure 4.13: Distribution of rate limits for all ASes propagated top-down

viable. Further, a user may be allowed to use more than their limit until the AS's overall limit is reached. Then, users who had exceeded their limits would be evicted from caches first.

An interesting question is how rate limits may propagate through the system. That is, if a tier 1 provider limits its tier 2 customer at rate x, and the tier 2 node limits its tier 3 customer at rate $f(x)$, what is the expected usable rate of an end-client? To understand this, we evaluated how rates propagate in our avoidance routing simulator. This simulator will be discussed in detail in Section 5.2.

We propagate rates with two different approaches. The first approach is to have each tier 1 AS set its max limit. Then, it splits this limit between its peers and customers. When a node has heard from all its providers, it splits a portion of the rate received to each of its peers. When a node has heard from all its peers, it adds all the rates received from providers and peers together, and splits it to customers. This continues to propagate

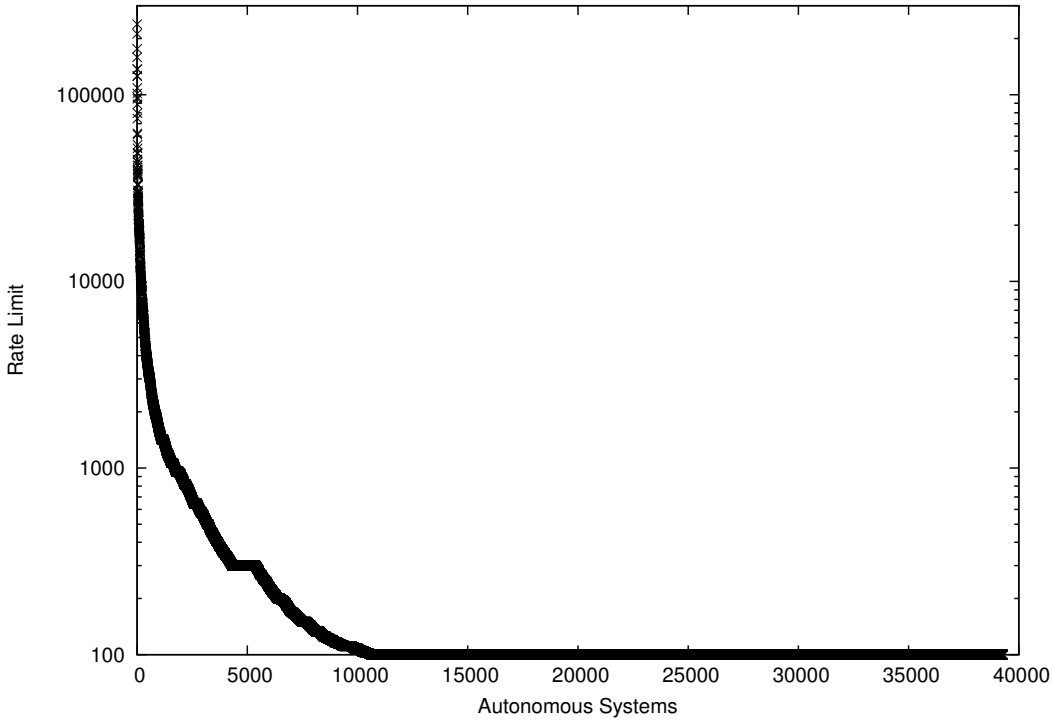

Figure 4.14: Distribution of rate limits for all ASes propagated bottom up

until all nodes have received some rate. Using this mechanism, we limit tier 1 providers to a 100,000 rate and set a hard minimum limit of 100. Thus, every node will have at least a 100 rate.

Figure 4.13 shows the distribution of rates among all ASes. This distribution is what we expected based on the power-law distribution of the Internet [FFF99]. Tier 1 ASes dominate the Internet graph by being highly connected, while the majority of ASes on the Internet have only one or two neighbors. Because of this, tier 1's will split their rates among all their neighbors, so each customer of a tier 1 will only get a small portion of the tier 1 rate. This will continue to propagate down until stub ASes are only provided with small amounts of rate.

This gives us an understanding of how quickly rate decreases from a top-down approach. How much rate would the core require if each stub AS asks for a specific rate and that rate propagates up? This is the second mechanism we tested to propagate rates. In this case,

Table 4.1: Increase of maximum and average rate limit as minimum grows

Minimum Growth	Maximum Rate Increase	Average Rate Increase
50	104,789	156

each stub AS requests a 100 rate, which it splits among its providers. As each AS hears from all its customers, it takes the total requested rate and split its among its peers and providers. This ends when all core ASes have heard from all their customers. Figure 4.14 shows the distribution of rate when pushed bottom-up. This distribution represents the power-law dynamic of the Internet, but also shows that the required core rate to satisfy all requests is 250,000 rather than 100,000.

Table 4.1 shows how the maximum required rate and average rate increases as stub ASes request more rate. If all stubs request 50 more rate, the most highly connected AS requires an increase in allowable rate of 104,789 to be able to satisfy the increased requests. Further, the average rate increases by 156. If each stub AS requests a 1,000 rate, the most highly connected core node would need to be able to satisfy 2.1 million requests.

Since we know how rate propagates and grows, we can now estimate how many malicious users are required to overwhelm the system. In this example, the core sets its maximum rate at 100,000, and all nodes have a minimum rate of 100. Since we know that the core needs a 250,000 available rate to satisfy a stub rate of 100, stubs will be able to overwhelm the core in this setup. How many stubs are required in order to pull this off?

To understand the capabilities of a misbehaving AS, we allow ASes to misbehave in three ways. The first way is to have each stub AS pick a destination and try to use its full rate towards that destination. The second way is to have each stub pick as many destinations as it has rate and use only a rate of one for each destination. In this way, each node spreads its rate through the network. The third way is to have stubs pick paths that all transit the same core AS.

We can see in Figure 4.15 that the maximum rate available drops rather steadily until

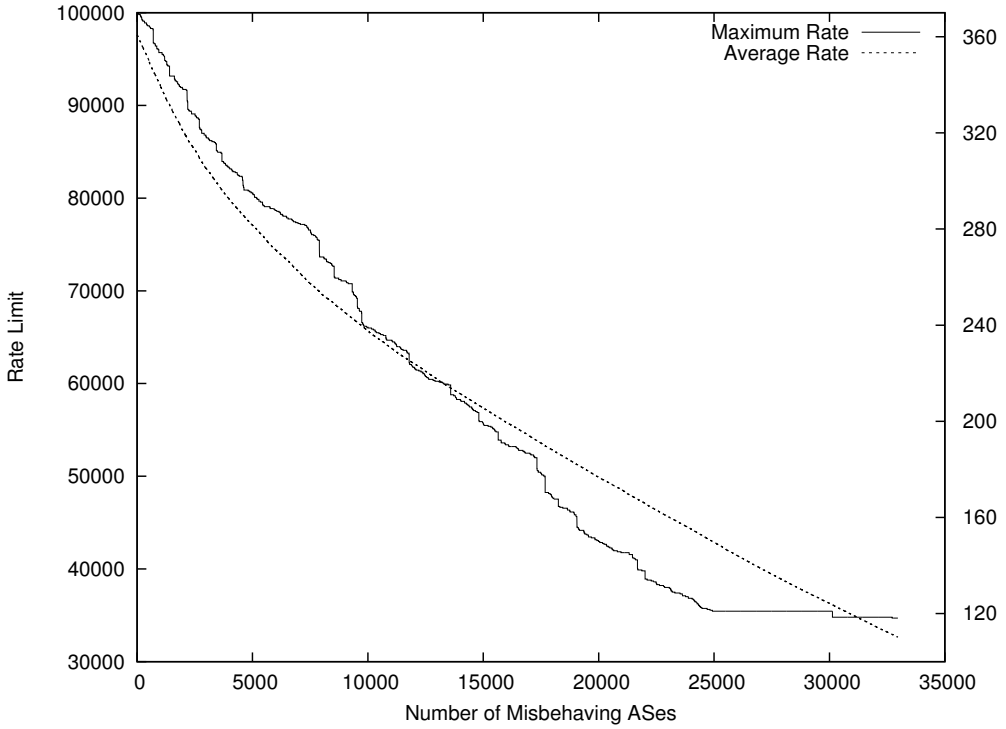

Figure 4.15: Maximum and average available rate with full-rate attack

reaching a rate of about 35,000. The jitter in the drop is caused by the random choice of destination. Some destinations may not require paths that transit core nodes. The 35,000 minimum occurs by stubs reaching dead ends. That is, when a node sends out a request, any of the nodes that it tries to use may have already reached a 0 rate. Normal avoidance routing would continue to search regardless of this case. For this experiment, if a dead end is reached, we give up. The average rate in the system, represented by the right axis, decreases rather monotonically.

Figure 4.16 shows the maximum and average rate available if the attacker uses the "spread attack." This attack acts worse then the targeted attack, since the attacker does not put all his "eggs in one basket." If an attack fails, it is likely that the rest of the attacks may succeed. Further, more nodes in general will be affected by the spread nature of the attack, thus the average rate decreases further.

The final attack is the focused attack. In this attack, all stub ASes pick paths that

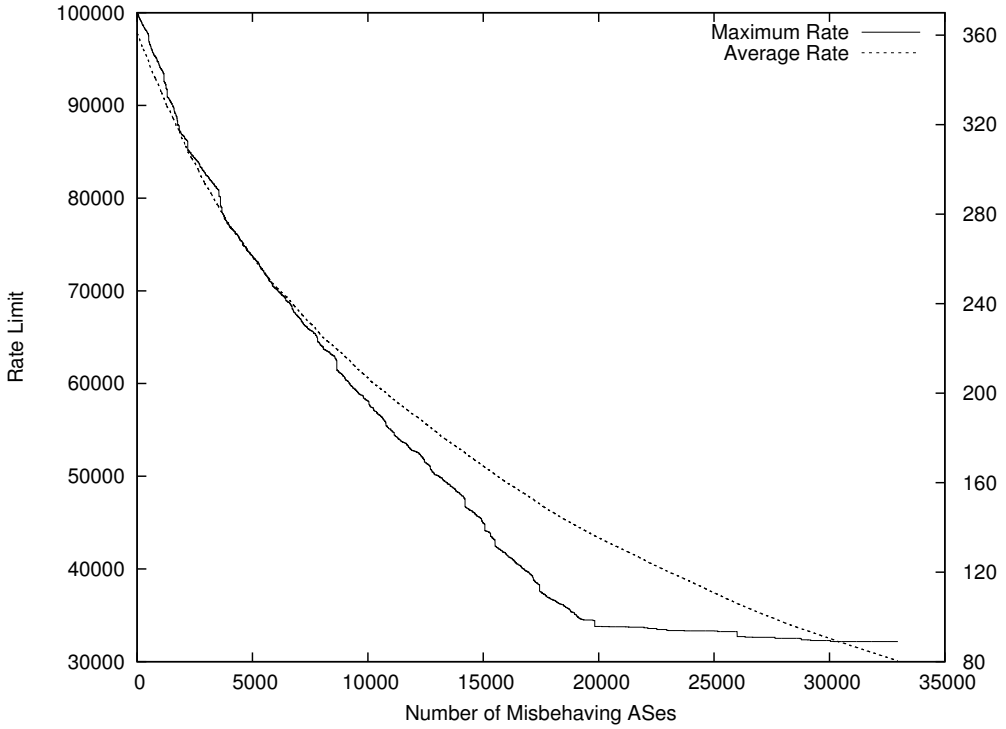

Figure 4.16: Maximum and average available rate with the spread Attack

transit the most highly connected AS. Using this attack, the target AS should lose all its rate rather quickly. Figure 4.17 shows the effect of this attack. The targeted attack quickly exhausts the core AS's rate, requiring only 500 stub ASes to succeed. The effect of the full-rate and spread untargeted attacks on the targeted AS are also shown. While these attacks do not try to target this AS, since this is the most highly connected AS, it is likely to get hit by these attacks. In fact, it takes only $\tilde{4}200$ stubs in the spread attack and $\tilde{8}100$ stubs in the full-rate attack to reduce the target to a 0 rate.

While it may seem that the number of ASes required to overwhelm a core AS is small, in fact this may not be true. This assumes that each attacker will have access to the full rate of its ASes. We can assume that each stub will continue the rate limiting to its customers, thus each individual machine will only have access to a limited amount of rate. For example, if each machine has only 1% of its providers' rate, then it would take 50,000 machines to effectively pull off the targeted attack. This attack also makes several

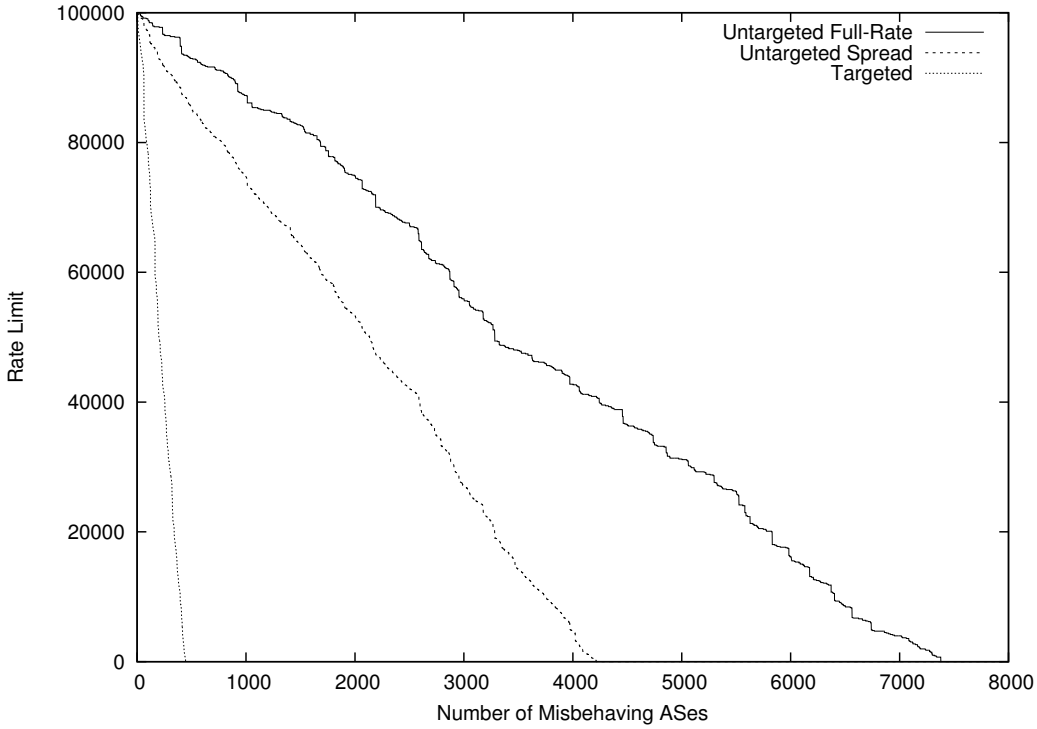

Figure 4.17: Effect on the targeted AS during the different attacks

assumptions regarding the maximum rate available to core nodes. Finally, to succeed in this attack, an attacker needs to create very specific avoidance requests that may require far more topological knowledge than is available to the attacker.

4.6.3 Avoidance Message Spoofing

There is another attack vector open to adversaries that is not addressed by rate limiting. What if an adversary sends false success or failure messages? This could prematurely terminate a search when the search should not have been terminated. Further, depending on the action taken, this could result in the adversary actually receiving the data when they were not supposed to.

Avoidance routing has several mechanisms to prevent this attack. Success messages originate from the destination and are destined for the sender. For an attack to work, the attacker has to know both the sender and receiver of the original avoidance request.

Further, for both success and failure messages, the attacker has to know either the hash of the security criteria, or the security criteria itself. Since security criteria are variable in length, it is impractical to generate hashes for any combination. For added protection, we can replace the general hash of the criteria with a hash of the source, destination, criteria, and IP ID field. The ID field is generally chosen at random. For an attacker to spoof the hash, he would have to generate and send all combinations, which is impractical and easy to detect.

The final protection is that success and failure messages must arrive on the interface currently being tested. If the message comes from elsewhere, the message is treated as spoofed and ignored. This is a valid mechanism as the current neighbor conducting the search should be the only one who knows to send you (this router) the success or failure message.

What if the currently tested neighbor decides to send success or failure without actually conducting the search? We do not care if the neighbor sends failure. This would result in us not using this neighbor, which is the right choice since this neighbor is not behaving correctly. It is possible that this allows neighbors to "pretend" to use avoidance routing but never actually allow it. It shouldn't be too hard to detect this and take remedial actions. As we discussed in the rate-limiting section, we expect neighbors to negotiate the use of avoidance routing as part of a peering agreement. If one side is violating their contractual obligations, legal action may be possible.

Thus, sending false failure messages is not a huge concern, but what about false success messages? A node could claim they have a route to the destination even if they do not. How to exactly solve this problem is still open. The approach we use is to have the destination sign the success message which the sender validates. If the signature is invalid, the sender sends an avoidance message indicating the failure and requests that the path be rebuilt. Thus, any router who receives the request again will not resue the choice made previously. This may result in poor route generation, but will not result in the wrong person receiving

the data.

There are other security questions that have not been answered in this discussion. What happens if nodes lie about the security criteria? What if nodes decide to incorrectly forward packets regardless of avoidance routing? These questions are discussed in Chapter 6 and Chapter 7.

4.7 Usability

Making avoidance routing usable is necessary in order to get the system adopted. It would be foolish to have a user craft the avoidance requests or mark every packet by hand. Instead, these capabilities are built into the system, allowing the user to be able to use avoidance routing without it being a burden. New functionality is added to an operating system to allow user-level software to use avoidance routing.

Avoidance routing facilities are created to support UDP and TCP packets. Three primary system calls are required. The first, ar_connect(), creates the avoidance path. This function works the same as the standard connect call except it also takes a pointer to the avoidance criteria as well as the length of the criteria. This function will create the avoidance path and modify the sock structure to store the identifier for the avoidance path. In the case of TCP, it is smart enough to first build the avoidance path and then send the TCP SYN to build the TCP connection.

The other two calls are ar_send() and ar_sendto() which correspond to the TCP and UDP send calls respectively. These calls work the same as send and sendto except they add the avoidance routing option to the IP header and place the avoidance identifier into the header. This allows software to use avoidance paths for its data. Further, software has the option to use either send or ar_send.

There is an interesting question of what happens on the receiver side. It's possible that most receivers will be legacy receivers, i.e. receivers that do not understand avoidance

Table 4.2: Standard functions and avoidance counterparts

	Standard Function	Avoidance Function
Open Connection	connect()	ar_connect()
Send on Connection	send()	ar_send()
Send Generically	sendto()	ar_sendto()
Listen	listen()	ar_listen()

routing. In these cases, the last hop (their provider) would have to handle processing the avoidance request and sending of success. A facility to allow a receiver to understand avoidance routing should be provided as well. In this case, we have ar_listen(), which works the same way as listen() except it understands avoidance routing. It can be configured to accept all avoidance connections, or it can be set up with its own avoidance criteria. When a request is received, its criteria will be unioned with the receiver's criteria and a new avoidance request will be sent from the receiver. In this way, both the sender's and receiver's criteria will be matched. New receiving functions are not necessary, as once the data has arrived, avoidance routing is irrelevant. Table 4.2 shows the old transport kernel functions and the new avoidance routing versions.

While this new interface allows new or modified applications to use avoidance routing, it would excellent if we could support legacy applications. This can be accomplished via a SOCKS proxy [LGL96]. These proxies are commonly used to allow legacy applications like Firefox to use modern services like Tor without complete rewrites. SOCKS proxy can be run both locally or remotely. In this case, the proxy would be run on the local machine. The proxy would accept standard connections and turn them into avoidance connections. Some applications already have SOCKS support and can simply be directed to the proxy. Other applications would have to be serendipitously redirected to the proxy using firewall rules or other kernel support. Figure 4.18 demonstrates how this would work at a high level.

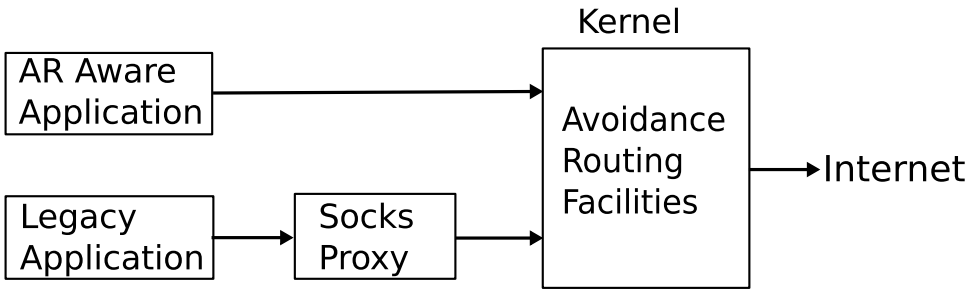

Figure 4.18: Interfaces for legacy and modern applications

4.8 Partial Deployment

Avoidance routing would, of course, benefit from total deployment. But clearly, avoidance routing will not be deployed all at once. Further, total deployment is not likely to happen as some players will never participate. Avoidance routing can work in partial deployment, provided that deployment is contiguous. The deployment must be contiguous so that a trusted avoidance route between source and destination can be discovered. ASes that do not deploy the system must be recognized and treated as untrustworthy.

Further, it may be possible to support non-contiguous partial deployment if a secure method of transit between one island of deployment to another is used. For example, if an anonymity network is deployed between islands, this can be used to protect data transiting between them. However, these methods are generally easily detectable, and may be attacked as a way to cut off one island from another.

4.9 Intra-AS Avoidance Routing

Intra-AS routing is the routing within a single AS rather than the routing between ASes. Not all nodes within an AS will have the same properties. For example, an AS may have a diverse geopolitical distribution [GR02]. Avoidance routing treats the AS as having all properties of its router. Due to this, it is possible that avoidance routing does not need an

intra-AS routing solution. If all the nodes within an AS are trusted, controlling the routing within the AS is not necessary. However, if any of the nodes are untrusted, the default avoidance routing design will not use this AS at all. For our primary design, this limitation is valid.

A future optimization is to make avoidance routing a trusted router mechanism, rather than a trusted AS mechanism. In this way, ASes that have untrustworthy properties could still be used, as long as the routing within the AS avoids the untrusted nodes. This requires a different level of trust than currently exists in avoidance routing, but future deployments and trust agreements could make this trust possible.

Clearly, this future avoidance routing system must have a scheme for intra-AS routing. However, since the nodes are all under the control of a single authority, intra-AS routing is generally simpler then inter-AS routing. The main protocols for intra-AS routing (called interior gateway protocols (IGP)) are "Open Shortest Path First" (OSPF) [Moy98] and "Intermediate System to Intermediate System" (IS-IS) [ISI02]. Both protocols are link-state. This means that both protocols have a complete understanding of their network's topology. To perform avoidance routing, one simply removes nodes that do not meet the appropriate criteria and then performs Dijkstra's algorithm. In this way, a path to the next BGP router can be found.

Of course the interior routers will need to know the security properties of their neighbors. This can be obtained in the same manner as in BGP, by annotating routing advertisements. While some changes will need to be made based on which protocol is modified, this clearly can be done.

We developed a version of avoidance routing called link-state avoidance routing (LSAR). LSAR has complete topological knowledge of the network it is traversing, as all link-state protocols do. As described above, to find the path from source to destination, we simply remove all invalid nodes and run Dijkstra's algorithm. Since link-state uses a breadth-first search, the path found will always be optimal. This is an advantage over the paths found

by our distributed DFS.

A limitation of our current LSAR implementation is that each node must run Dijkstra's algorithm. It is possible for just the first node to receive the request to run Dijkstra, and simply inform all further nodes within the topology of the path. We have not make this optimization, but it is clearly something that could be done rather cheaply.

4.10 Avoidance vs. Allowance

A final question is, why avoidance routing? Why specify criteria we do not trust? Why use a blacklist? Instead, why not specify the criteria we do trust? We could simply use a white-list. Fundamentally, these two ideas are the same. If the set of all properties in S and G is the set of properties we distrust, then G' is the set of properties we trust. We could as easily route on G as G'.

In fact, avoidance routing could be trivially modified to perform "allowance" routing. Rather than sending out an avoidance request with criteria to avoid, the user would send out criteria to exclusively use. The same security properties would be used to perform allowance routing. However, instead of checking for a path that does not violate the criteria, we would check for a path where all nodes satisfy the criteria. If we found such a path, we could use it immediately. For our implementation, this would require only a minor change in logic. Further, the extensibility of avoidance routing message types allows us to easily add this feature.

For our design and evaluation, we chose to stick with avoidance routing over allowance routing. In principle, the same results we get for avoidance routing apply to allowance routing. So why choose one over the other? You may distrust only a few things and trust many things. On the flip-side, you may trust few things and distrust many things. In a real-world deployment, both systems could be deployed. A user would choose one over the other based on trust or distrust sets.

CHAPTER 5

Evaluation of the Avoidance Routing Search

Several aspects of avoidance routing need to be evaluated to understand how well avoidance routing performs. Perhaps the most important question is, given the structure of the Internet, is avoidance routing even possible? If we select only a few criteria and the entire Internet graph becomes disconnected, avoidance routing would be completely infeasible. Another important question is how long are avoidance routes? Are they significantly longer than the standard routes? How do they compare versus the oracular avoidance path? Finally, how often does the standard route actually avoid the properties one cares about? That is, how often is the avoidance route the standard route advertised by BGP?

Since avoidance routing uses a search mechanism, it is possible for a search to visit far more nodes than the actual path length. The more extraneous nodes visited, the more resources that are wasted. How many extra nodes are visited on a search? This leads into the final question — how much overhead does a node experience to conduct a search? All of these questions will be answered in the following sections, along with a discussion and explanation of some of the interesting and unexpected behaviors learned about both the Internet and avoidance routing.

5.1 Software Implementation

To test and evaluate avoidance routing, we implemented it into the Click Modular Router [KMC00], a software router that allows developers a easy way to manipulate and route packets. Click uses "elements", modular software which takes an action on a packet before

passing it along. A configuration file specifies how "elements" are connected. For example, one may have a "FromDevice" element, which grabs packets off the physical interface, connected to a counter element which simply counts packets it sees. This element, in turn, could be connected to a router element which determines how to forward the packet. Further, developers can create their own elements to modify, destroy, create or route packets.

We developed a Click element called the Avoidance Router. This router acts as a standard BGP router for all non-avoidance routing packets. When a packet arrives, it simply looks up the forwarding interface based on a hash of the destination. Unfortunately, Click does not currently have a BGP implementation. There are many reasons not to implement BGP into Click, but the most important one is that BGP relies on TCP. Click, as a router, lives at the IP layer and does not understand TCP. Thus, to implement BGP, one would also have to implement TCP into click. To avoid this, we use the Quagga Router Suite [Qua12].

Quagga is a collection of routing protocol implementations for end-user machines. Quagga manages the information used by these protocols as well as sending out relevant messages. However, Quagga does not forward packets. It relies on Linux to do this. We modified Quagga's BGP implementation to interface with our Click router rather than Linux. Click has a built in ProcFS interface to allow outside applications to interface with it. Whenever Quagga changes its route to a destination, it informs Click of the new route via the ProcFS interface. This way, Quagga handles BGP processing but our Click router handles packet forwarding.

As one might recall, avoidance routing augments BGP advertisements with the security vector. We modified Quagga so that it could add and update the security vector. This is done in a similar manner to the path vector modifications. First, we added a new Quagga BGP configuration parameter, *sec_properties*. This parameter is attached to the neighbor parameter, which is used to specify information about neighbors. In this way, we can specify the security properties of a nodes neighbor. Whenever Quagga forwards a BGP

advertisement, it will add the properties of the previous hop to the security vector, as well as add the next hops properties as discussed in the previous Chapter. Finally, Quagga informs our Click router of all its currently stored advertisements as this information is required for conducting avoidance routing. This information is also shared via the ProcFS.

Now that our router has the relevant information, it can forward packets normally. The router is also aware of avoidance routing and how to handle avoidance routed packets. When an avoidance request arrives, the avoidance router will enter the search phase and begin the process of finding a path to the destination as described in Chapter 4. When a success message or failure message is received, the router will take the appropriate actions. Finally, the router will forward avoidance packets via the caching mechanism described in Section 4.2.1.

Given our implementation, we can attempt to evaluate how avoidance routing performs. This implementation allows us to determine what the storage cost of advertisements are. Further, we can determine if there is a forwarding penalty to either avoidance routed or non-avoidance route packets. Our implementation is designed to give us baseline measurements for any type of overhead the avoidance router might face.

5.2 Avoidance Routing Simulator

In order to evaluate avoidance routing at scale, we built an avoidance routing simulator along with the implementation in the Click Modular Router. While the Click implementation allows us to understand the overhead costs of avoidance routing, without thousands of physical machines available to us, it would be difficult to understand how avoidance routing operates at large scales. Instead, we build the Avoidance Routing Simulator.

The simulator operates in the following way. First, it loads in a connectivity map of the Autonomous Systems (ASes) of the Internet. This AS map servers as our base graph. Our map is obtained from UCLA's Internet Research Laboratory (IRL) which has done

research to determine the AS map and continues to update it [Ucl12] [OPW10].

Next, we annotate each link with the relationship between the ASes on each end of the link. These are the customer, provider, or peer relationships required to implement the "no valley" and "favor customer" policies. We learn these relationships from both IRL [Ucl12] and from Caida [Cai12]. These relationships are inferred from connectivity information based on techniques described by L. Gao [Gao01], L. Subramanian et al. [SAR02] and X. Dimitropoulos et al. [DKF07].

Once we have both the graph and the annotated links, we must annotate the links with security properties. In our experiments, we annotate our links in three ways. First, we annotate them uniformly at random. This may represent a security property that is relatively evenly distributed, such as firmware version, AS number or some other property we have not considered. The second mechanism is via a half-Gaussian distribution. In this mechanism, a few properties cover a lot of nodes, with the likelihood of having a property following a half-Gaussian distribution. This could represent something like corporate ownership, where few large corporations dominate the space but there are many small corporations.

The final mechanism we use is to load geopolitical information from a known database. We used information obtained from Team Cymru [Cym12b] which maps ASes to country. This information is known to have inaccuracies, as it is obtained from ownership information held by ARIN, AFRINIC, APNIC, etc... This is only a rough mapping, but gives us a general understanding of how avoidance routing may behave using geopolitical properties.

To run an experiment, we first select a set of properties to avoid. We then pick a source at random and find all the nodes reachable from that source given the avoidance criteria. Next, we pick a destination at random from the set of reachable nodes. The destination will propagate advertisements to all its neighbors, who in turn will propagate advertisements further out. We have the ability to turn on or off policy. If policy is disabled, the shortest known path is propagated to all neighbors. If policy is enabled, paths are selected by "no-

valley" and "favor customer". That is, customer paths are prioritized when picking a path to advertise, and a peer or provider path cannot be advertised to another peer or provider.

Once all nodes have settled on an ad, we can begin to calculate avoidance paths. We have built in the capability to find the oracular avoidance path, that is the best avoidance path between source and destination. This is done by running a modified weighted Dijkstra's algorithm [Dij59], and uses the same implementation as LSAR discussed in Section 4.9. If a neighbor is reached via a peer or provider, than that neighbor will not use any peers or providers. If the neighbor is reached by a customer, than any neighbor is available.

Finally, we can run avoidance routing. We simply begin a depth-first search through out the graph, picking neighbors based on the advertisements they gave. Depending on the heuristic used, neighbors are sorted and tried in order. When the destination is reached, we record both the path length and number of nodes tried.

One may notice that the oracular path does not follow the "favor customer" policy but only the "no-valley" policy. While we implemented the ability to "favor customers", that is, to use customer links prior to peers or providers, we ended up finding paths longer than those found by avoidance routing. This is when we discovered that an oracular breadth-first search cannot perfectly emulate a limited information depth-first search.

Recall that if a node has a valid advertised path to a destination, it uses it immediately. This is due to the fact that forwarding along a known path is better than searching on a unknown path. However, this may result in avoidance routing using a known provider path when a unknown customer path may exist. The oracle would discover this longer but unknown customer-only path, while avoidance routing would find the shorter but known provider path. Figure 5.1 shows a topology in which avoidance routing would find a different route than the oracle.

In this example, S wants to reach D avoiding gray. Again, upward links are provider links while downward links are customer links. Customer C has advertised to S the $\{C$ - I - $D\}$ route while provider P has advertised the $\{P$ - $D\}$ path. Standard routing would select

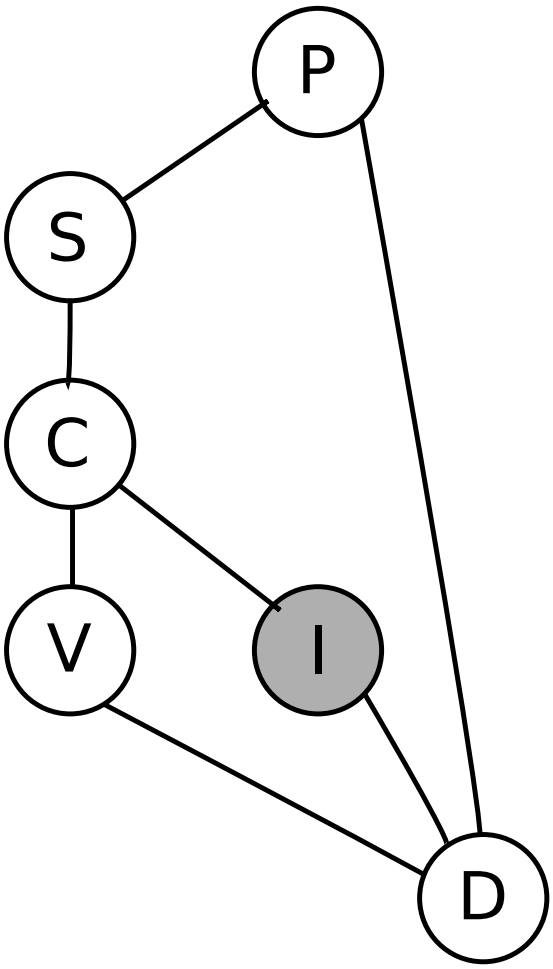

Figure 5.1: A simple topology where "favor customers" breaks

the customer path through C. However, since I is invalid, avoidance routing can either used the unknown $\{C$ - V - $D\}$ route or the known $\{P$ - $D\}$. Thus the oracle, following strict "no-valley" and "favor customer" rules and having no knowledge of what paths a node is aware of, will discover the longer $\{C$ - V - $D\}$. Avoidance routing, discovering a known valid path, will use the shorter provider-based $\{P$ - $D\}$ path. With a strict path length metric, the avoidance route will be better than the oracle. To avoid this problem, we implemented the oracle to find the shortest path, regardless of favoring customers or not.

The simulator is capable of a few other mechanisms besides simply calculating avoidance

paths. The simulator can also determine the connectivity of the graph after selecting avoidance criteria. Since selecting avoidance criteria is congruent to removing nodes from the graph, we can determine what the connectivity of the remaining graph is after the nodes are removed. Finally, the simulator can perform the rate-limiting experiments described in Section 4.6.2 by building the topology and relationships and then propagating rates.

5.3 Available Paths

How many avoidance paths exist? This question reduces to how much of the graph remains connected after we remove nodes from the graph? There has been significant work done in this area that shows that the graph remains highly connected and highly resilient to failures [ZZT05] [WZM07]. The results of this research is in line with the results we reached in the work done for the prospectus of this work.

While these results are promising, we felt that we would like to confirm this result given the AS graph we now are operating with. While previous work was thorough, it was done as late as 2007, which may not represent the Internet of today. Further, previous research focused on nodes failing based on past failure history, whereas nodes in avoidance routing can be removed for any reason, such as their geopolitical location. Finally, we wanted to take policy into account, as this effects how many paths are available between any two nodes.

Determining how many nodes remain connected after removing some nodes requires that these nodes are "removed". We do this by marking nodes as unusable in the avoidance routing simulator. How we mark these nodes is based on one of four distributions — uniformly at random, half-Gaussian, geopolitical from largest Internet presence to smallest, and geopolitical from smallest to largest. We have explained why we choose these distributions in Section 5.2. The reason we do two runs of geopolitical location is because a few large countries dominate the Internet graph. We felt it would be interesting to know how

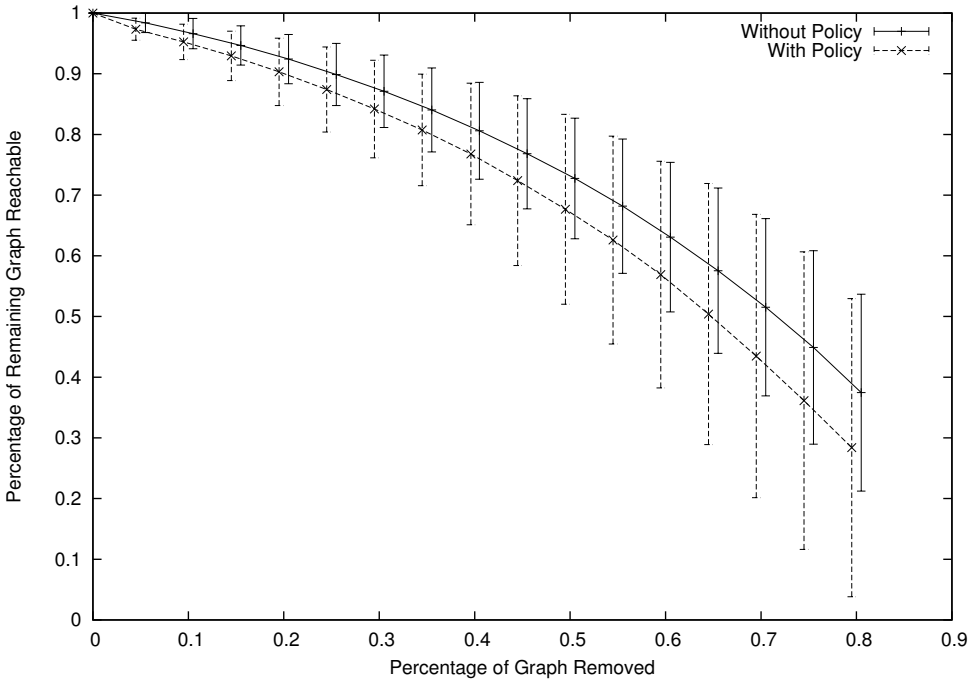

Figure 5.2: Connectivity of remaining graph after uniform removal

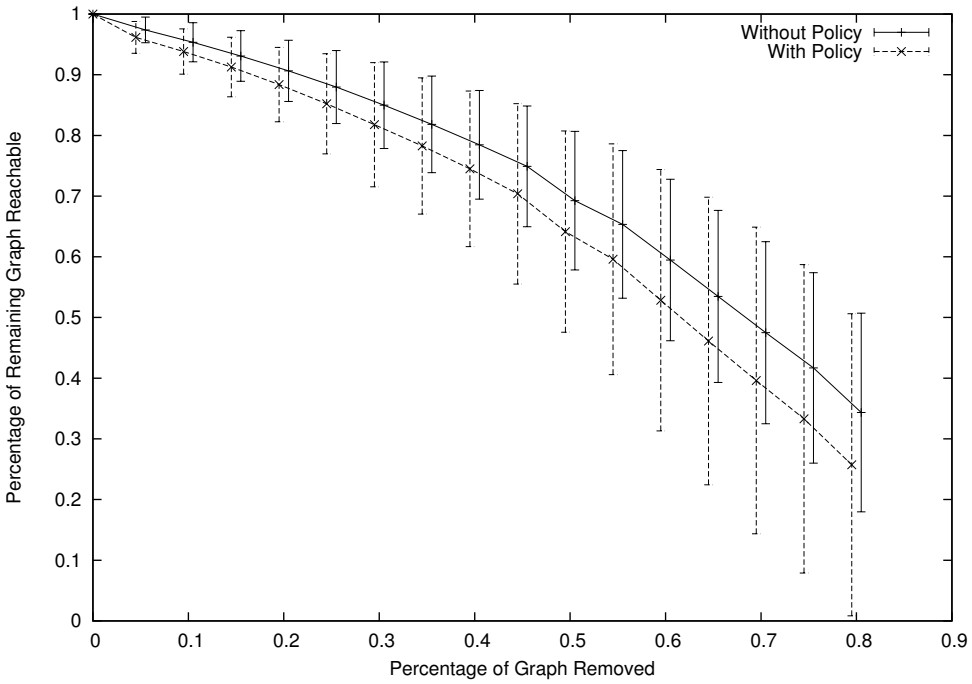

Figure 5.3: Connectivity of remaining graph after half-Gaussian removal

removing these countries would affect the graph as well as how removing small countries (countries that effect the fewest nodes) but leaving the large countries (countries that effect the most nodes) in would affect the graph.

In our uniform and half-Gaussian experiments, a node was given properties based on the distribution used. These properties consisted of a value between (1-100). After distributing properties, we removed nodes from the graph with a specific property. After removing 5% of the graph, we would run Dijkstra's algorithm to calculate the graph connectivity. We continued to remove the graph in 5% chunks until approximately 80% of the graph was removed. We called this entire process a trial, and each experiment consisted of 1200 trials. After running these experiments, we calculated error using three standard deviations.

Our geopolitical experiments are run a little differently. The primary difference is that the distributions are fixed. A node will always have the same property every time, corresponding to what country it is in. As stated in Section 5.2, the topology is obtained from UCLA's Internet Research Lab and the mapping of ASes to countries is obtained via data from Team Cymru. Our removal order is also fixed, either removing the country with the largest Internet presence first, or the smallest first. Due to this, no matter how many trials are run, the results will always be the same. Thus, there is no experimental error for these experiments. That does not mean there is no error! Both the topology and geopolitical mappings may have their own errors.

Figures 5.2 and 5.3 show the connectivity of the remaining nodes after a certain percentage of the graph has been removed. Connectivity is shown with and without policy. Note that the data points have been offset slightly for readability. In the uniform removal case, connectivity remains high; roughly 65% of nodes are still connected, with 55% of the graph removed. After this point, the graph begins to disconnect much more rapidly. Removing nodes based on a half-Gaussian distribution experiences a similar effect, remaining highly connected until roughly 60% of the nodes are removed.

The reason for this effect is rather simple and is based on the power-law of the Internet.

The AS graph is made up of a small, fully-connected core and a very large, sparsely-connected edge. In fact, the core is made up of only a few hundred nodes while the graph itself has approximately forty thousand. When removing nodes at random, either uniformly or with a half-Gaussian distribution, it is more likely that a sparsely-connected edge node is removed. As long as the core remains intact and most nodes can reach the core, then most of the graph will remain connected. It is only when a tipping point is reached, when most of the core has been disconnected, that the graph itself becomes disconnected. Its possible that during the selection process, you get unlucky (i.e. the core is highly selected), and this is shown by the lower edge of our three standard deviation error bars.

The primary difference between the policy and no-policy connectivity graphs is that edges around the outside of the graph are usable in the no-policy version. For example, in the no-valley policy, you are allowed to take at most one peer hop. That is, you cannot string together peer hops. However, without policy, you can string together as many peer hops as you would like. Further, "valleys" can be used as a valid transit mechanism, which would clearly violate the "no-valley" policy.

Figure 5.4 shows the effect of removing nodes based on geopolitical location, from largest to smallest. The first point on the graph occurs at approximately 35%. This occurs because the United States has the largest presence, containing 35% of all autonomous systems. The graph stays highly connected, even as large countries are removed. One would expect the graph to disconnect rapidly as large countries systematically remove the core. However, while the US represents 35% of the AS graph, it does not represent 35% of the core. This makes a large portion of the remaining graph still connected. Further, since all the nodes removed generate a connected sub-graph, and each country itself represents a connected sub-graph, the remaining countries should be connected. It is not until the majority of countries are removed that the graph becomes disconnected.

An interesting artifact of this graph is that connectivity goes **up** as more nodes are removed. There are two factors that cause this. The first is that we measure the connec-

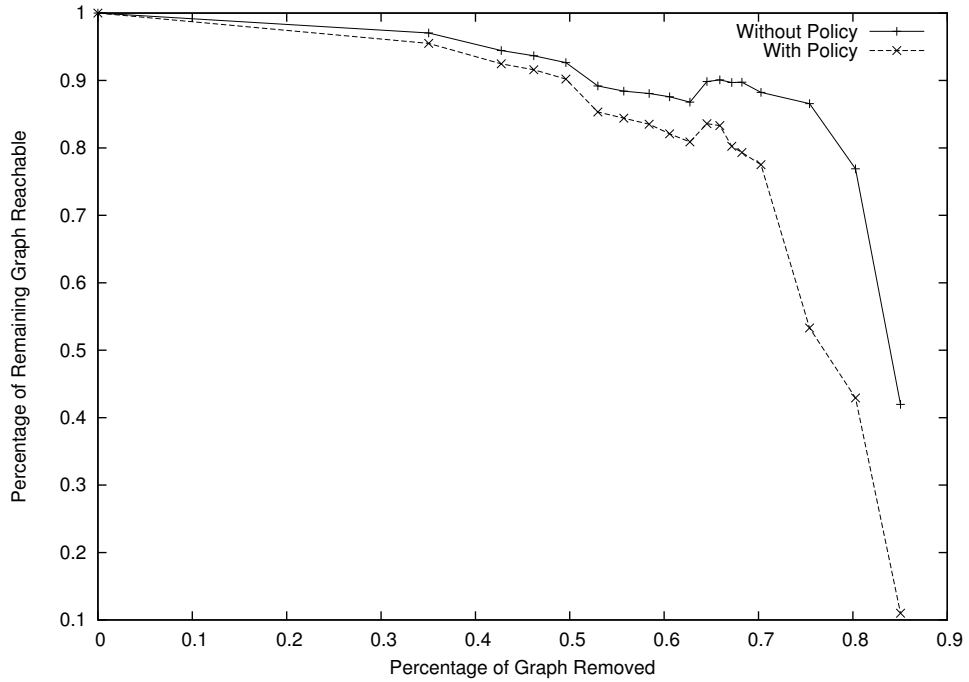

Figure 5.4: Connectivity of remaining graph after geopolitical removal from largest to smallest

tivity of the **remaining** nodes in the graph. That is, if we have removed 70% of the graph, what percentage of the remaining 30% of nodes are connected? The second factor is that countries tend to be contiguous regions. When a country is removed, it will tend to be self-contained. Further, if the removed country was already disconnected from the graph, the connectivity of the remaining nodes will go up. This is what is occurring here. Of the nodes removed at the 62% mark, most of these nodes are contained in a country that was already disconnected. Thus, after removing this country, the remaining nodes are more highly connected.

Once again, there is a difference between using and not using policy. The largest difference occurs as the graph begins to get disconnected, near the 70% mark. This occurs because at this point, the majority of the core is gone. While the lack of policy allows regions connected by valleys or peers to remain available, the policy version does not. With all of the large and medium-sized countries removed, there is no way to go from one

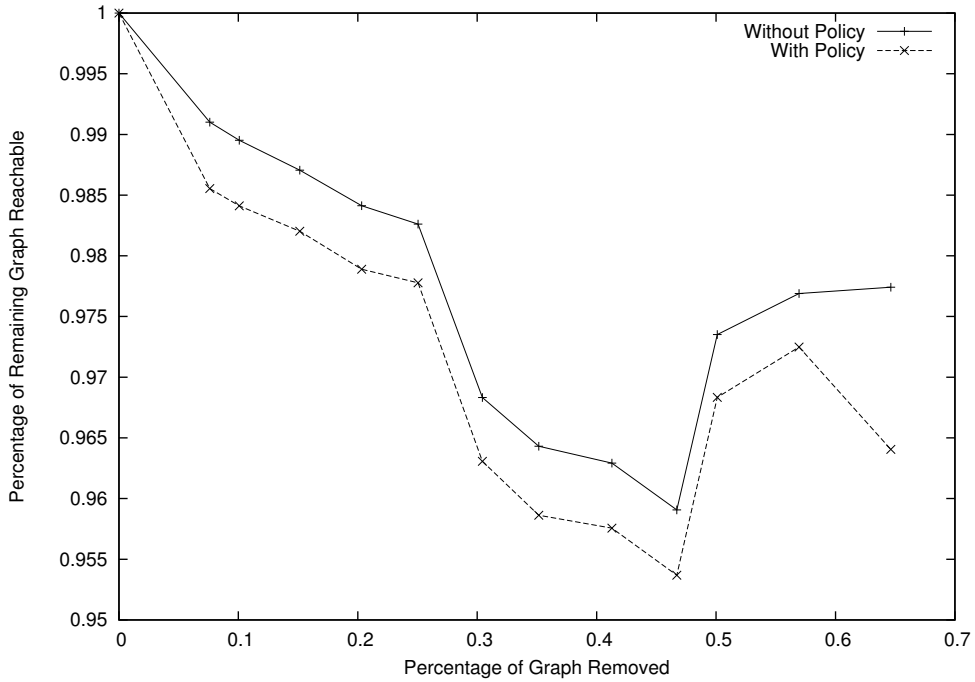

Figure 5.5: Connectivity of remaining graph after geopolitical removal from smallest to largest

pocket to another.

The last removal mechanism, removing geopolitical properties from smallest to largest, is shown in Figure 5.5. In this case, the graph remains highly connected the entire time, never dropping below 95%. This result should be obvious. All the largest countries are highly connected to the core. Even with everything removed but the United States, since the United States is connected to itself, the graph remains highly connected. Finally, connectivity increases here for the same reason as previously. When removing a disconnected region, the remaining nodes are more connected.

5.4 Path Length

When removing nodes from the Internet graph, it stands to reason that the path taken from a source to a destination may get longer. How much longer these paths become is

an important question. If the paths are exceptionally long, avoidance routing may not be very practical. Further, since avoidance routing uses a depth-first search, it is possible that these paths may be not optimal. In fact, it is well known that depth-first search may result in exceedingly bad paths, and has been the subject of multiple research efforts [Kor85] [YG02] .

The heuristics discussed in Section 4.4 are designed to prune the search space and pick better paths. How well these heuristics work will be shown in this section. We conducted several experiments to understand how well avoidance routing and our heuristics work. These experiments show how much longer avoidance routing paths are when compared with the standard BGP route. We also show how much longer the optimal avoidance routing path is. Each experiment uses a different security property distribution mechanism. We experimented with the four different distributions discussed in the previous section. Finally, in order to understand the effect of policy on avoidance routing, we ran experiments with no policy, with the "no-valley" policy and with the "no-valley" plus "favor customers" policy.

In these experiments, we use the same basic strategy as our previous experiments. Nodes are removed based on their corresponding distribution . Next, a source and destination are picked at random from the nodes still connected. That is, the source and destination must be reachable. Finally, avoidance routing is run to find a path from the source to the destination. Avoidance routing is run multiple times, once for each heuristic. Again, we do each trial 1200 times for each removal level (5%, 10% etc...) and for each heuristic. Thus, one experiment consists of 102,000 runs of avoidance routing with 20,400 different source/destination combinations. Error is calculated using three standard deviations. For readability, some of the graphs will leave off error, but error will be shown and discussed later in the Section.

Figure 5.6 shows the increase in path length for all five of our heuristics using a uniform property distribution. The increase in path length using a half-Gaussian property distribution is shown in Figure 5.7. In both cases, BGP is being simulated with no policy.

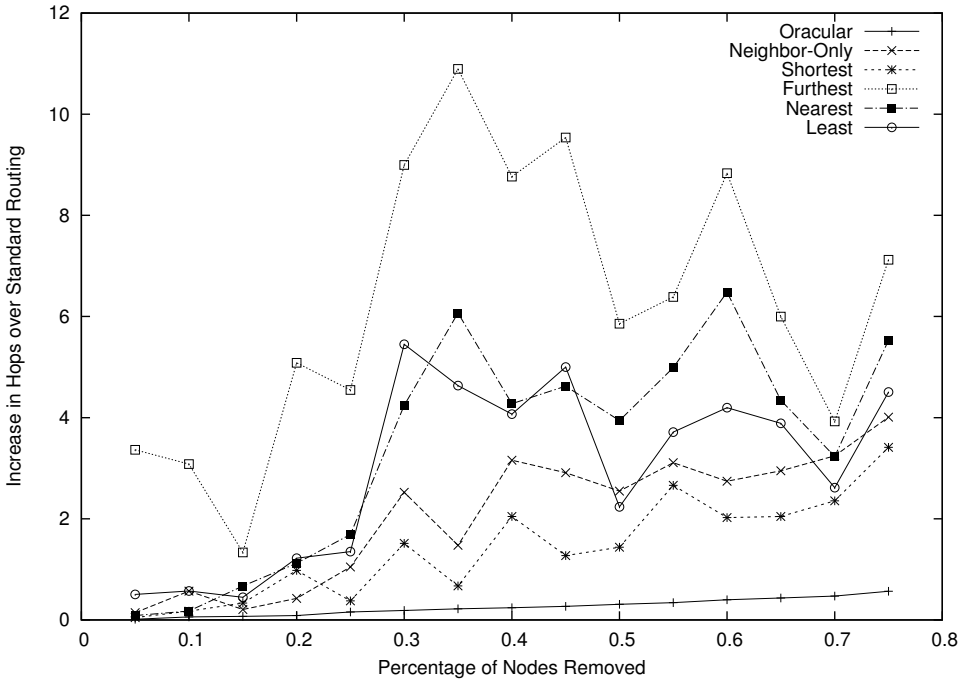

Figure 5.6: Increase in path length with no policy using uniform properties

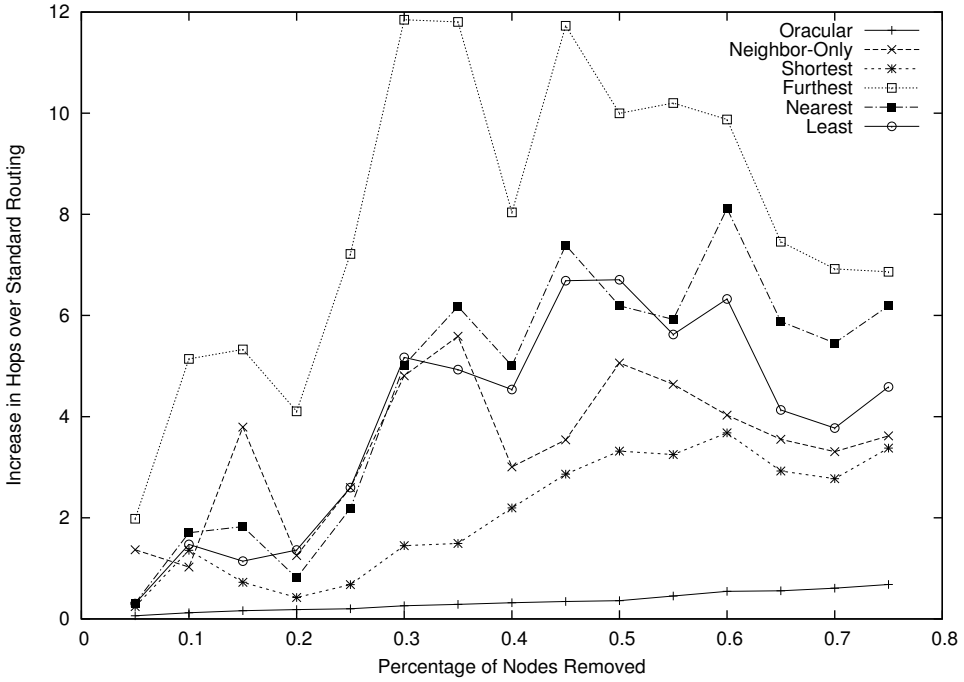

Figure 5.7: Increase in path length with no policy using half-Gaussian properties

There are several interesting things to note about these results. The first is that several heuristics perform worse than neighbor-only avoidance routing. Furthest collision performs consistently the worst. This should not be particularly surprising. This heuristic favors paths where the collision is further away. Longer paths have a greater opportunity for further collisions, thus longer paths will be favored.

However, both nearest collision and least total collisions also tend to perform worse than neighbor-only. For nearest collision, this is the result of search constantly heading towards collisions rather than the destination. The search will favor paths with a nearby collision, even if the path is significantly longer. Least total collisions also experiences this same bias, except for paths that have a lower collision count. In fact, the best heuristic is shortest-path first. Since our metric is path length, a heuristic that favors short paths is clearly the best.

Neighbor-only performs well because it picks neighbors based on BGP ordering (i.e. the order BGP would advertise paths to its neighbors). In the no policy case, this ordering is shortest-path first, thus neighbor-only is basically using this heuristics. Why doesn't neighbor-only perform as well as SPF? This is because neighbor-only does not know about completely valid paths and thus may have to "wander" to find the path.

Figures 5.8 and 5.9 show the increase in path length for the geopolitical property distribution. Figure 5.8 shows effect of removing properties in order from largest to smallest while Figure 5.9 shows the effect of the reverse ordering. We experience similar results to that of the uniform and half-Gaussian removal patterns. In both of these cases, the furthest collision heuristic tends to perform the worst, while SPF performs the best.

There is an important distinction between these two results, which is similar to that shown in the discussion regarding connectivity. In Figure 5.8, large countries, and thus the majority of the core, are removed first. Because of this, the path length to reach a destination quickly grows. By contrast, removing small countries first barely affects the path length. The spikes shown in Figure 5.9 are barely blips when compared to the other

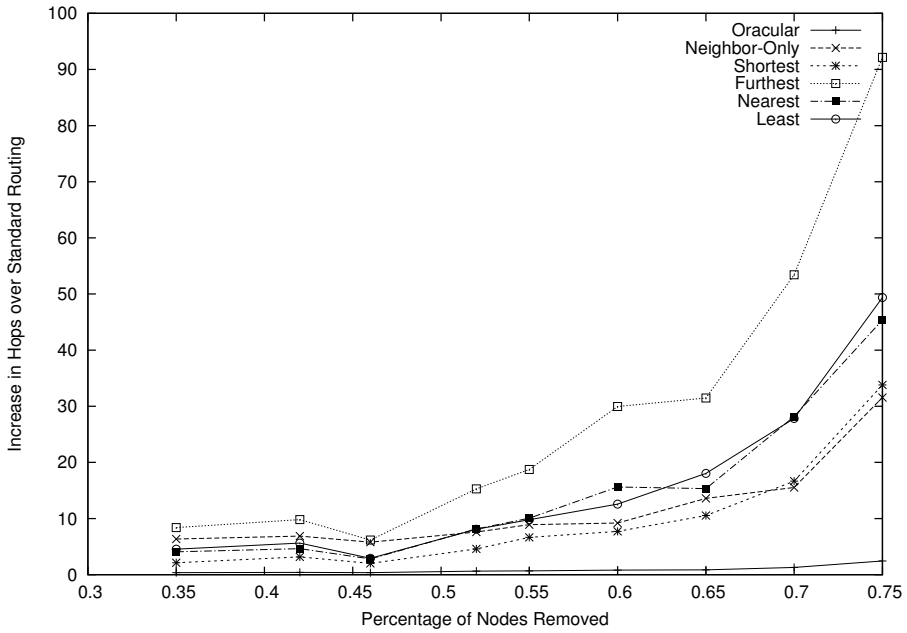

Figure 5.8: Increase in path length with no policy using geopolitical properties from largest to smallest

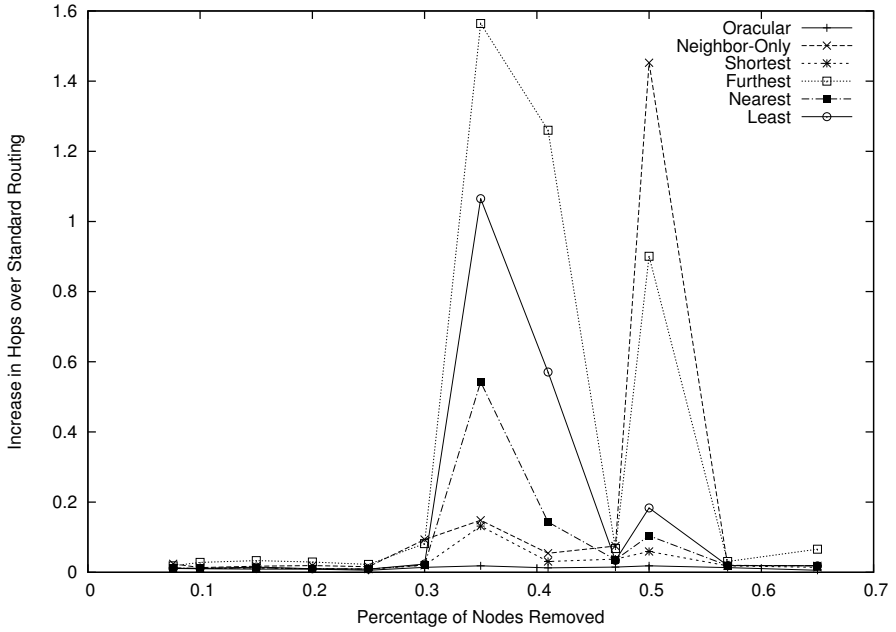

Figure 5.9: Increase in path length with no policy using geopolitical properties from smallest to largest

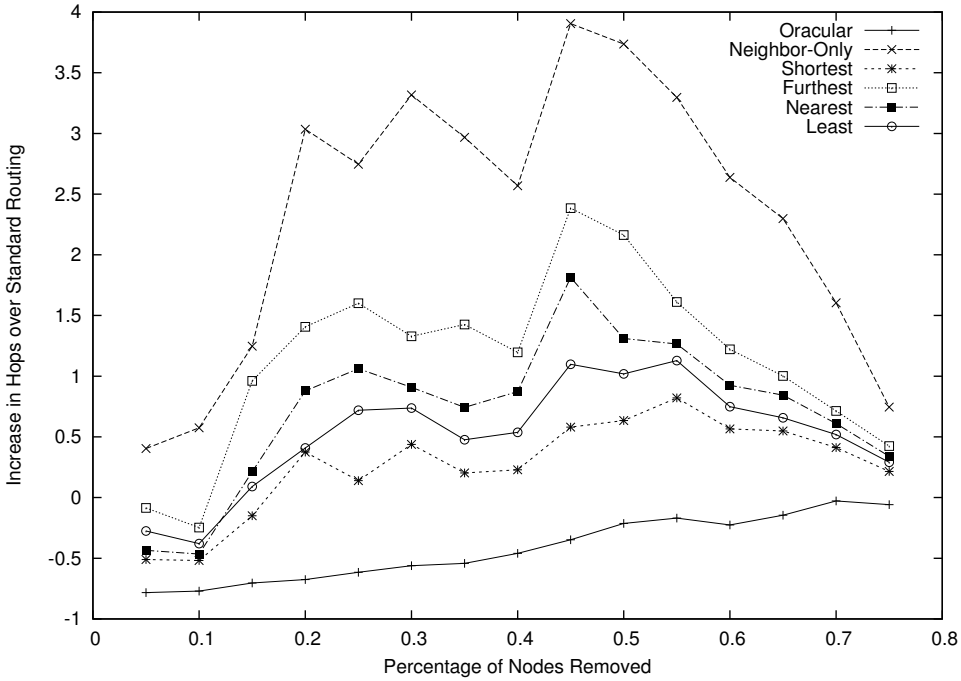

Figure 5.10: Increase in path length with policy using uniform properties

figure. For those curious, these spikes are caused by the randomness introduced by picking sources and destinations.

Clearly, shortest-path first is the best heuristic when policy is not considered. What happens when we introduce the "no-valley" policy? Figures 5.10 and 5.11 show the effect of policy on the uniform and half-Gaussian distributions respectively. Several fundamental changes have occurred with the introduction of policy. First, neighbor-only is generally the worst heuristic. The ordering amongst the remaining heuristics remains unchanged, with furthest collision being the worst and SPF being the best. The second interesting change is that avoidance routes may be **shorter** than the standard BGP route. Why both of these effects occurred will be discussed in Section 5.6 which includes some insights shown in Section 5.5 about total nodes visited.

We see the same effect occur when considering geopolitical properties. In Figure 5.12 we can once again see that neighbor-only is the worst and that some avoidance paths are shorter than the standard BGP path. This is also true in the smallest to largest removal

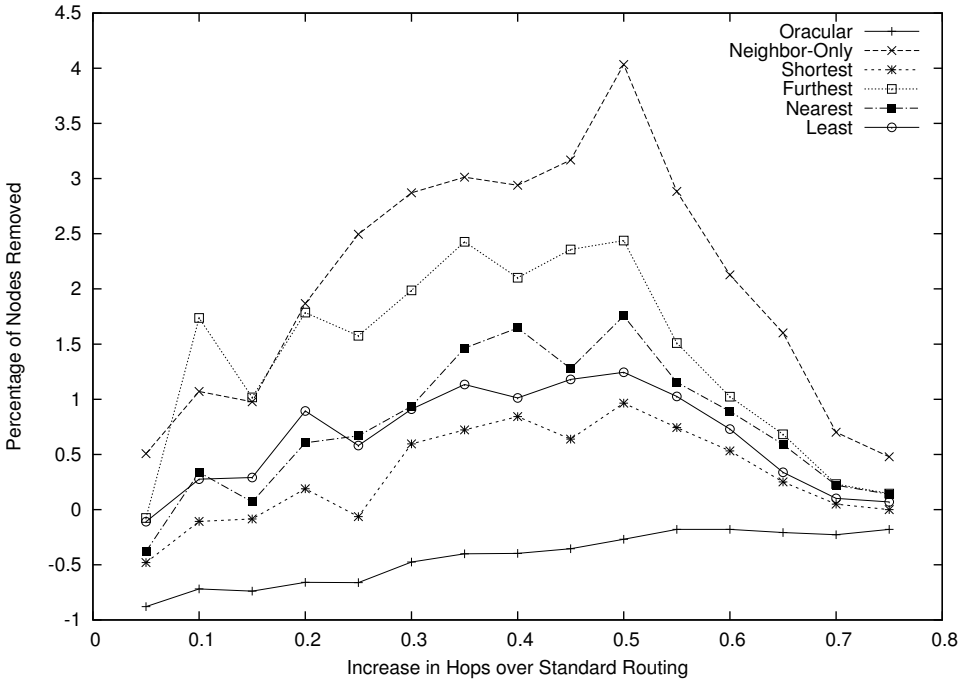

Figure 5.11: Increase in path length with policy using half-Gaussian properties

strategy shown in Figure 5.13, except that, once again, overall path length increase is very small. The increase is also smaller than that of routing without policy. One would expect that since path selection is restricted by policy, avoidance paths would be longer. While this is true, the standard BGP path is also longer. It should be clear by now that removing all the small countries first has almost no effect on the Internet graph.

Shortest-path first is now clearly the best heuristic when policy is not considered as well as when using the "no-valley" policy. The only remaining policy element to consider is the "favor customers" policy. Recall that this portion of current policy says that an AS will prefer to use a customer route to a destination over a peer or a provider. This changes the ordering that paths are tried.

In the no policy version, paths are selected simply by how well they match the heuristic being used. The "no-valley" selection process restricts the available paths to be used by the neighbors relationship to that AS (i.e. customers and peers can only use customers, providers can use anyone). However, valid paths, those that satisfy "no-valley" are sorted

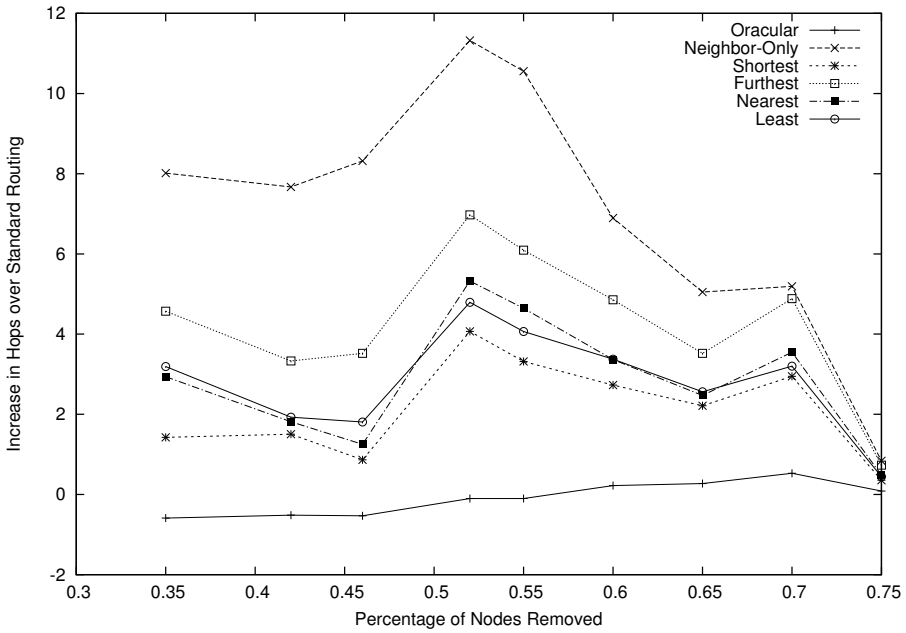

Figure 5.12: Increase in path length with policy using geopolitical properties from largest to smallest

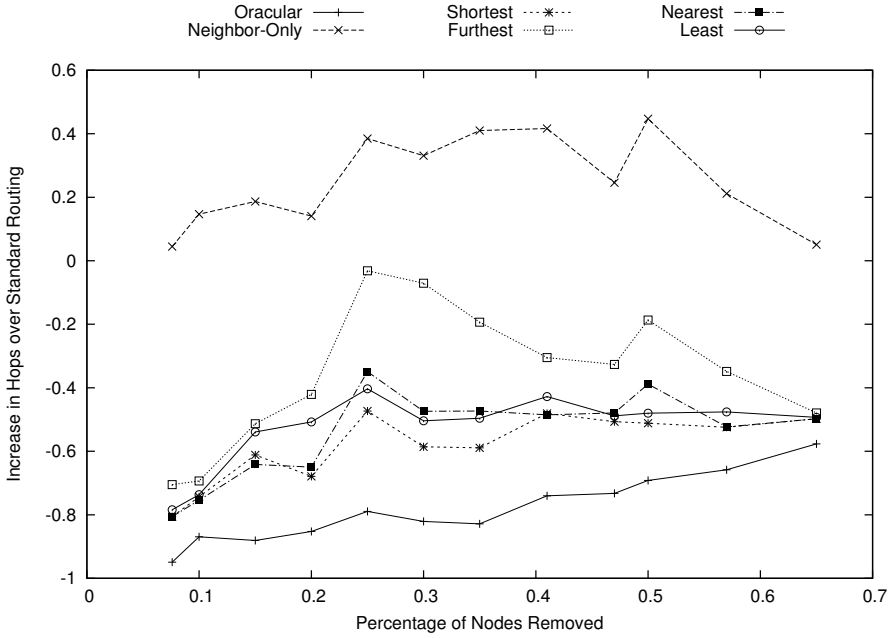

Figure 5.13: Increase in path length with policy using geopolitical properties from smallest to largest

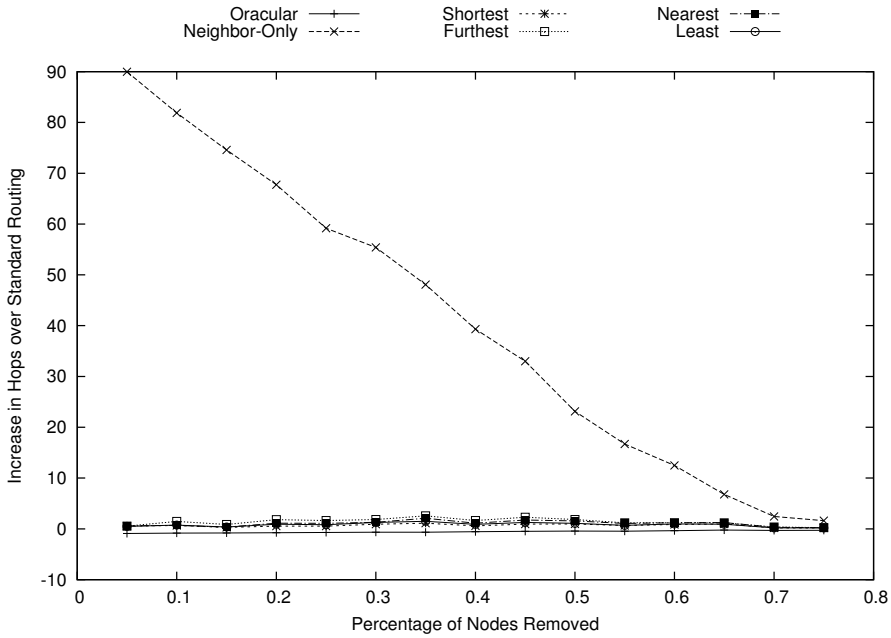

Figure 5.14: Increase in path length for uniform properties when favoring customers

based on the heuristic just like the no policy version. In "favor customers", paths are first restricted based on "no-valley" and then sorted by neighbor relationship. Thus, all customer paths will appear before any peer or provider paths. Finally, each subgroup will be sorted by heuristic. For example, if the heuristic is SPF, the shortest customer path will be the one tested first. Further, all customer paths will be tested before the shortest peer path, regardless of length.

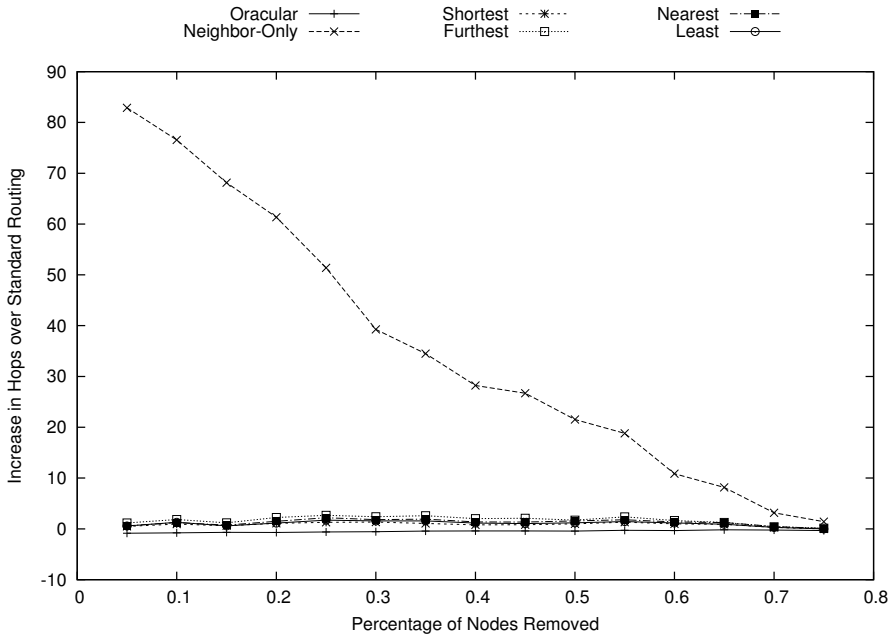

Figure 5.15: Increase in path length for half-Gaussian properties when favoring customers

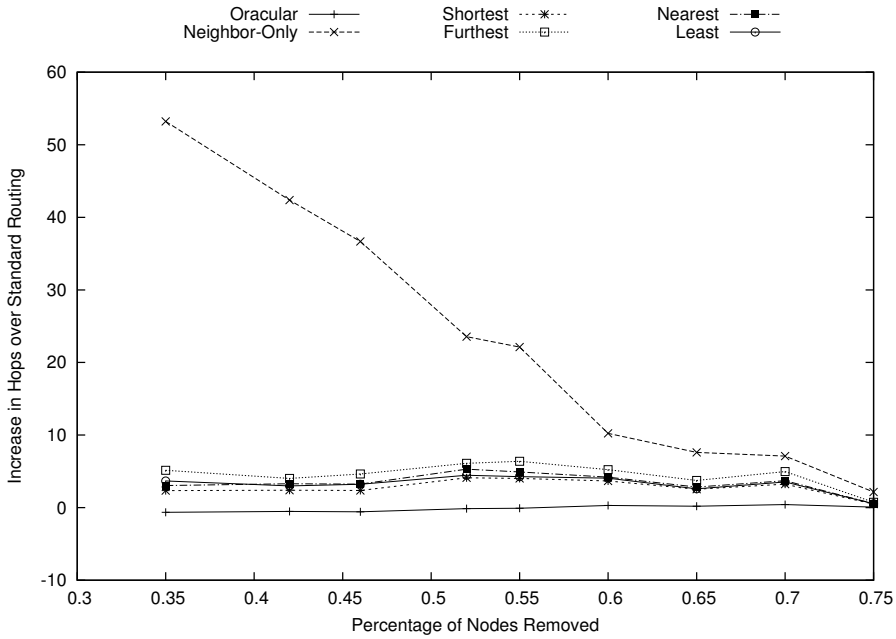

Figure 5.16: Increase in path length with policy using geopolitical properties from largest to smallest while favoring customers

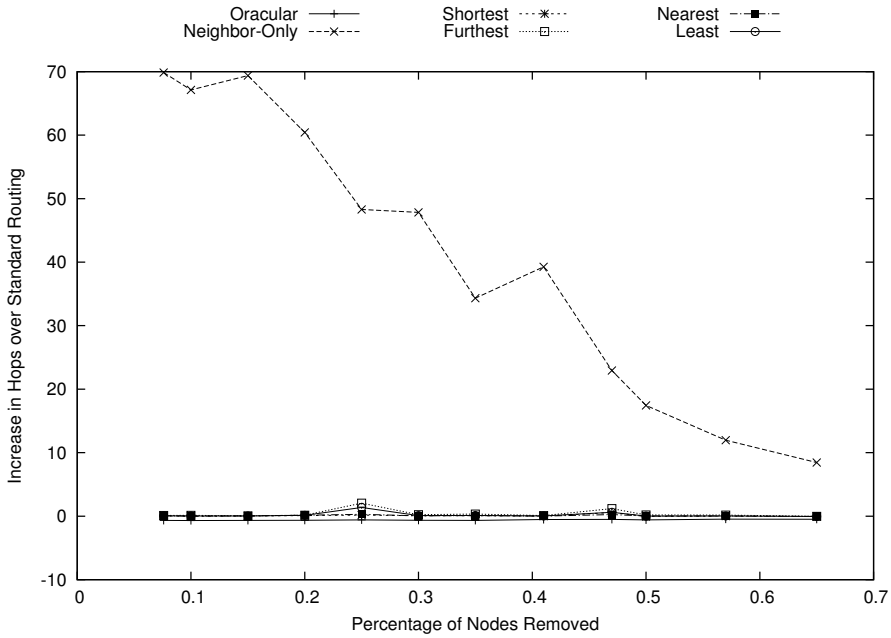

Figure 5.17: Increase in path length with policy using geopolitical properties from smallest to largest while favoring customers

Figures 5.14, 5.15, 5.16 and 5.17 show the effect of the "favor customer" policy on all four distributions. These graphs have one important take away. The increase in path length for neighbor-only is greatly increased. These paths are so long they dominate the rest of the figure, making any of the other heuristics unreadable. Why this occurs will also be discussed in Section 5.6.

Removing neighbor only allows us to have a better understanding of how "favor customers" affects avoidance routing. An example of this is shown in Figure 5.18. The simple conclusion to take from this is avoidance routing is not affected by the "favor customer" policy. In fact, all four distributions bear this out. There is generally no difference in the average increase in path length. Again, why this occurs will be discussed Section 5.6.

While there has been no difference in average path length, what about maximum? In fact, our entire discussion up to this point has dealt with the average path length. What is the distribution of path lengths? How bad can the paths get?

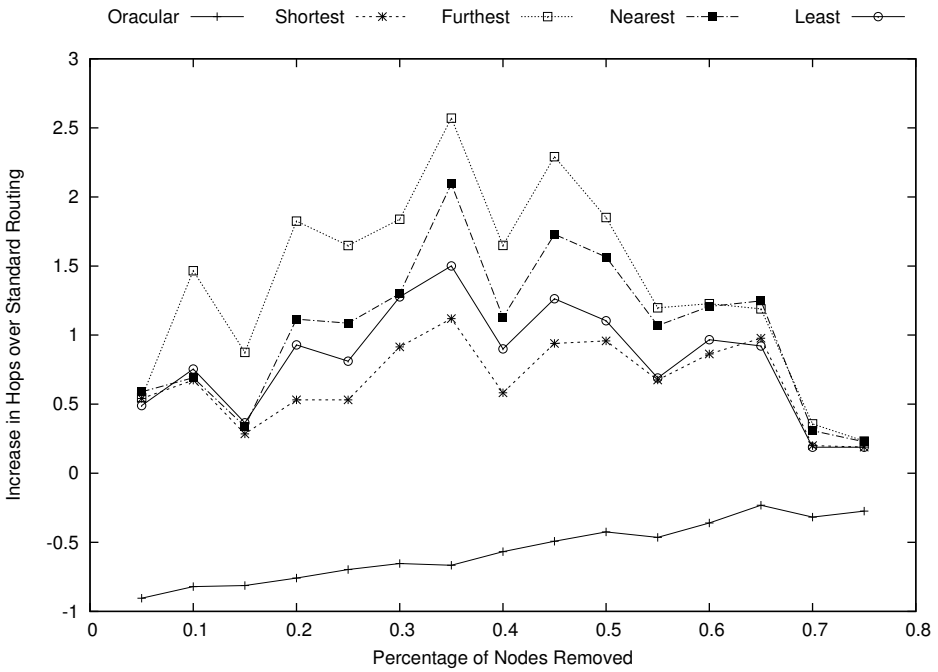

Figure 5.18: Increase in path length for a uniform property distribution when favoring customer

Figures 5.19, 5.21 and 5.23 show the neighbor-only and SPF heuristics for uniform property distribution and Figures 5.20, 5.22 and 5.24 show the results for the geopolitical "largest first" distribution. Figures 5.19 and 5.20 represent the increase in path length without policy. The increase with the "no-valley" policy is shown in Figures 5.21 and 5.22 while Figures 5.23 and 5.24 display the "favor customers" policy. For ease of reading, the other distributions have been removed, especially since SPF tends to be the best heuristic. Further, the data points have been offset to make the error bars easier to read. These error bars represent three standard deviations.

These graphs show that the paths can be significantly longer than the average. In the case of no policy, both SPF and neighbor-only have error bars that are above 80 nodes. The geographic distribution can have significant path lengths of nearly 200 hops. When policy is introduced, SPF performs well, rarely having paths above 40 hops in the geopolitical distribution and above 30 in the uniform distribution. As we had already experienced,

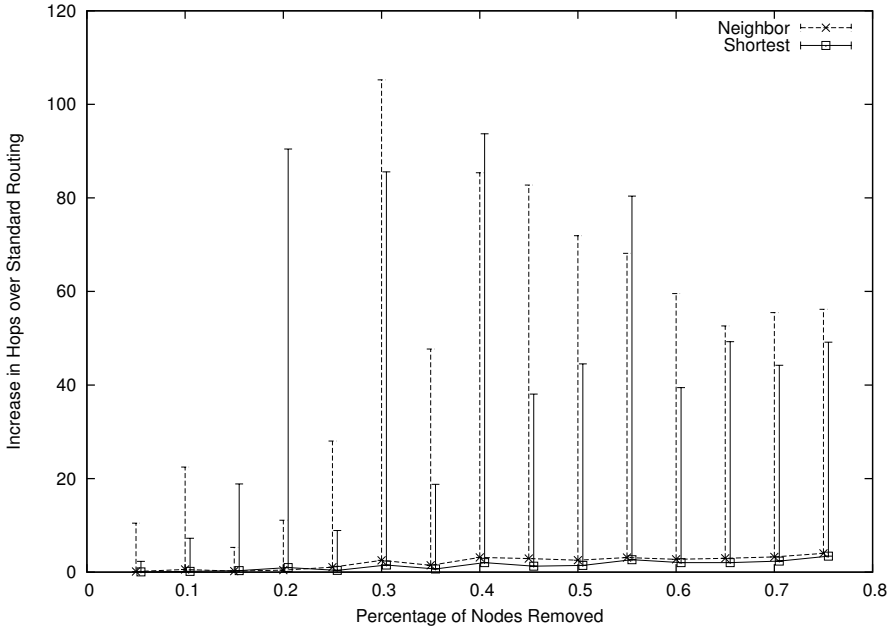

Figure 5.19: Increase in path length and error with no policy using uniform properties

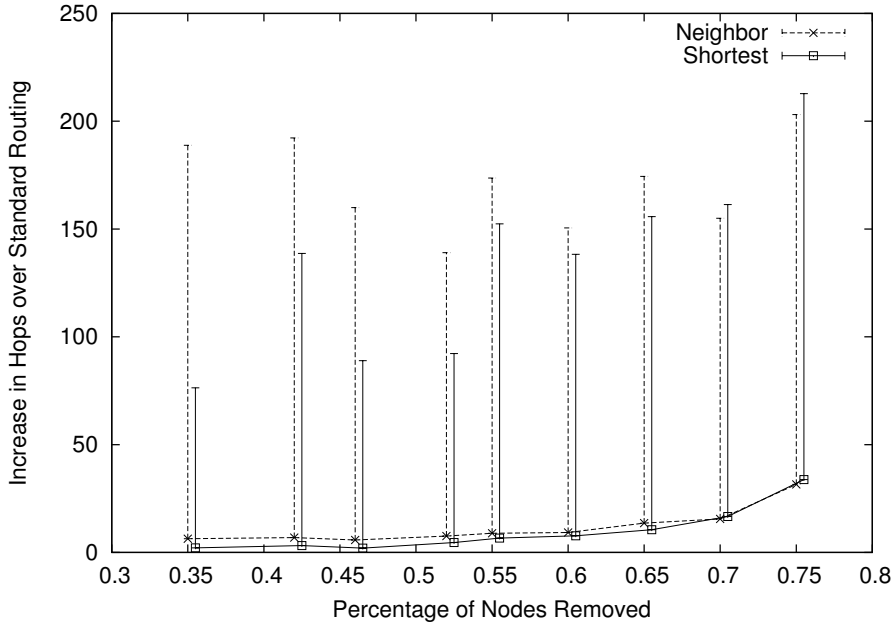

Figure 5.20: Increase in path length and error with no policy using geopolitical properties

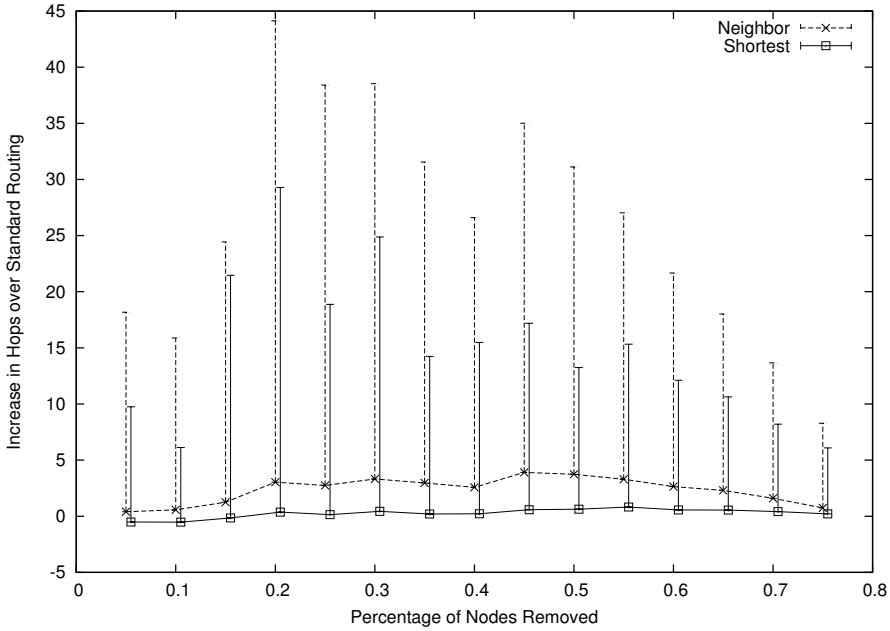

Figure 5.21: Increase in path length and error with no-valley policy using uniform properties

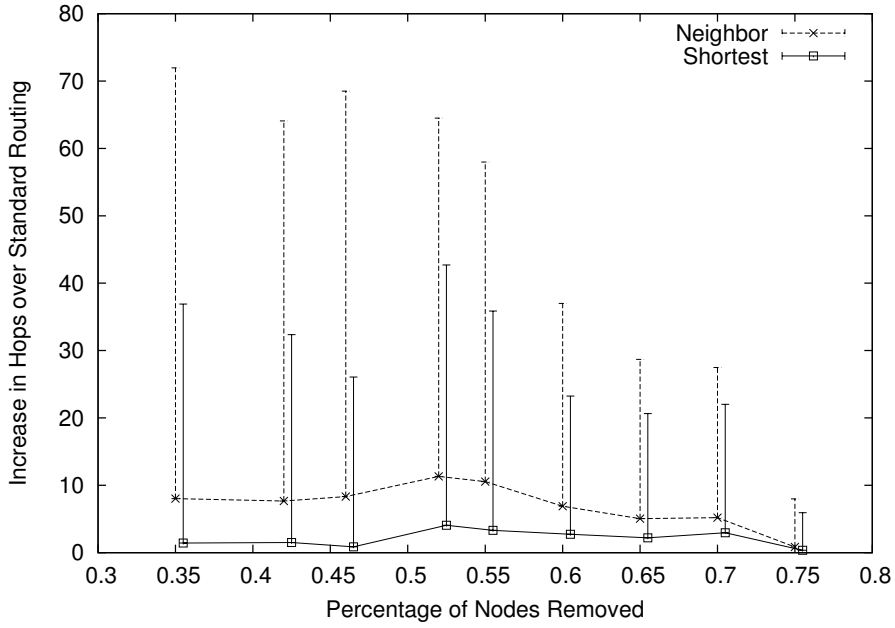

Figure 5.22: Increase in path length and error with no-valley policy using geopolitical properties

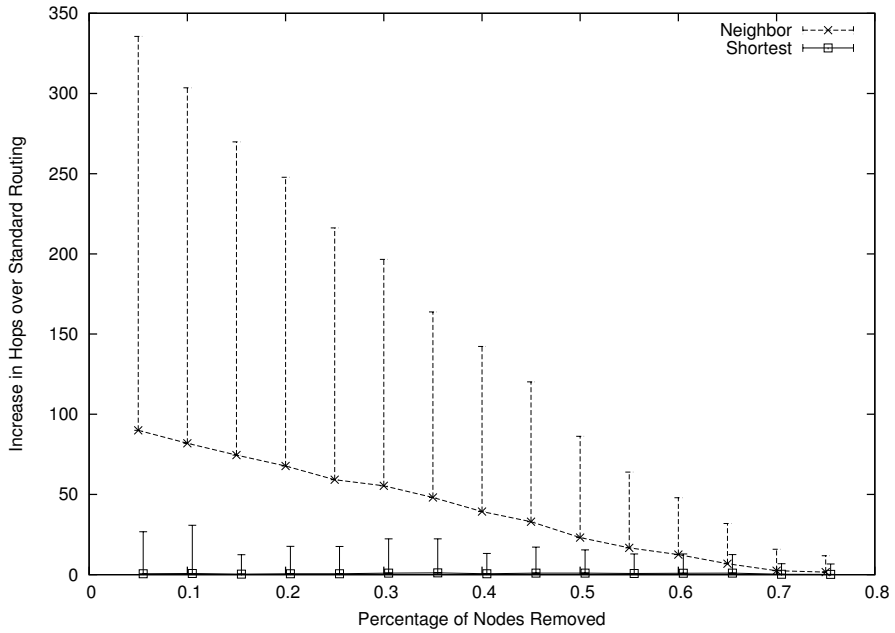

Figure 5.23: Increase in path length and error with favor customer policy using uniform properties

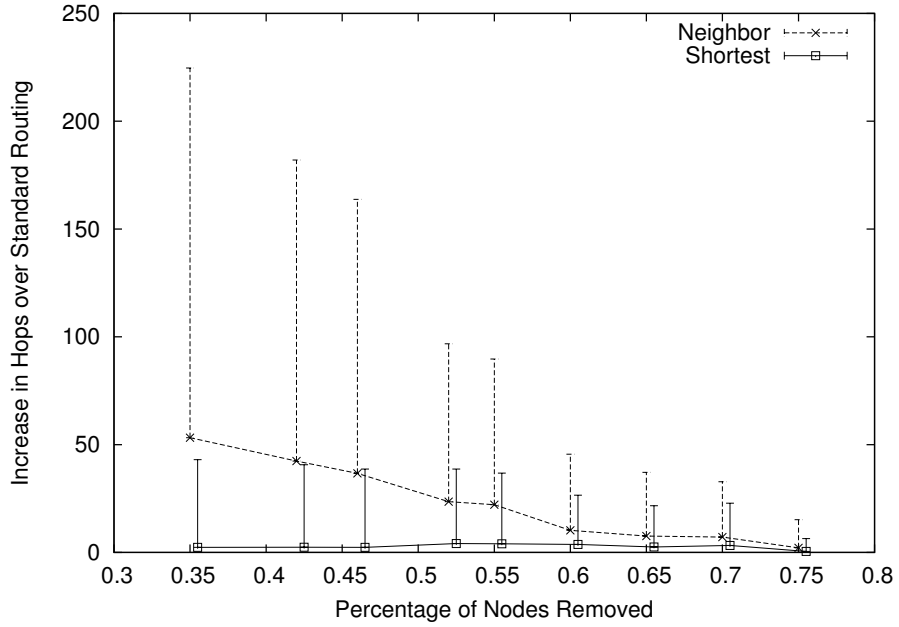

Figure 5.24: Increase in path length and error with favor customer policy using geopolitical properties

Table 5.1: Maximum observed path for different property distributions (Percent of graph removed)

Policy	Heuristic	Uniform	Half-Gaussian	Geo (Largest)	Geo (Smallest)
No Policy	Neighbor	1020 (30%)	1330 (5%)	1069 (35%)	847 (50%)
	SPF	1037 (20%)	1300 (10%)	1166 (46%)	128 (35%)
No Valley	Neighbor	157 (20%)	171 (5%)	171 (35%)	173 (15%)
	SPF	186 (5%)	172 (5%)	153 (35%)	144 (15%)
Favor Customers	Neighbor	289 (5%)	255 (5%)	210 (35%)	503 (47%)
	SPF	205 (10%)	193 (10%)	165 (35%)	685 (47%)

neighbor-only does especially poorly when the "favor customer" policy is used, having paths as long as 300 nodes.

The maximum observed path for all four property distributions given neighbor-only and SPF are shown in Table 5.1. It is not surprising that incredibly long paths may exist when there is no policy. Since there is no "structure" to a no policy Internet graph, it is quite possible to "snake" through the graph. One may also notice that the majority of longest paths occur when most of the graph is still connected. Since more of the graph is connected, it is more likely to take a longer path than when most of the graph is disconnected. For example, when 50% of the graph is removed, the maximum possible path from a source to a destination may be only 100 nodes. However, with only 5% of the graph removed, the maximum possible path may be as many as 1000 nodes.

We do see some long paths when large portions of the graph are removed. The 685 or 503 node paths shown for the geopolitical (smallest) distribution reflect this fact. This occurs, partly, by making a bad decision at the beginning of the search. Since most of the graph is removed, going a bad way may force the search to take a far longer path than going the right way. For example, in the 685 node path shown, the oracular path from the source to destination is only 6 hops. Due to the nature of depth-first search, this poor decision cannot be undone.

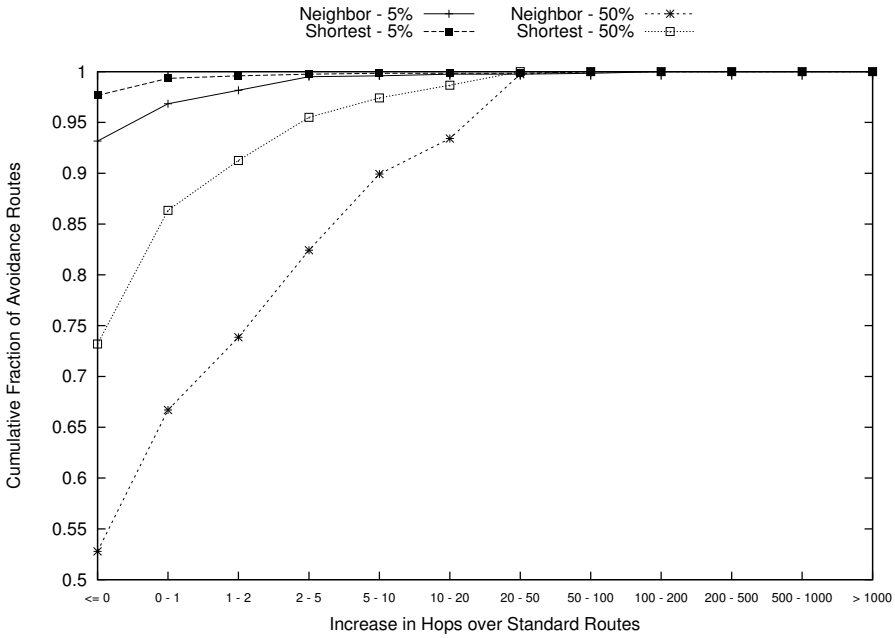

Figure 5.25: CDF of path length increase using uniform property distribution and "no-valley" policy

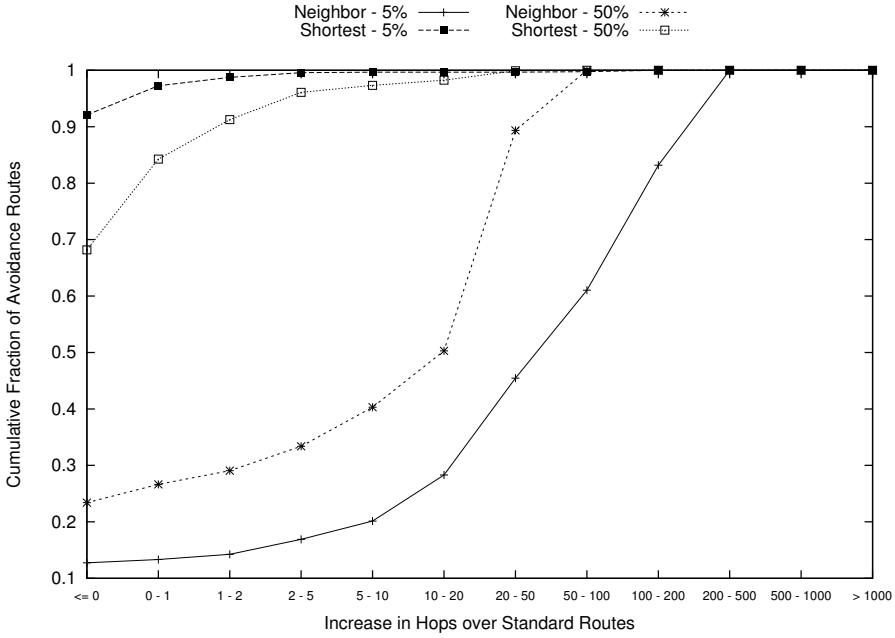

Figure 5.26: CDF of path length increase using uniform property distribution and favoring customers

We have seen based on maximum path and standard deviations, that paths can be exceptionally long. The final question is, how many paths are bad? Clearly, the average path length is low, thus a majority of paths must be short. But what is the distribution of these path lengths?

Figures 5.25 and 5.26 show the cumulative distribution function (CDF) for the uniform property distribution with "no-valley" and "favor customers" respectively. Again, we only compare the neighbor-only and SPF heuristics as SPF has been the best heuristic we've evaluated, and neighbor-only provides additional privacy guarantees. We also compare each mechanism when 5% and 50% of the graph has been removed to give a plausible upper and lower range for avoidance routing.

These CDFs highlight two important results. When 5% of the graph has been removed, we find that SPF finds paths that are no more than 2 hops longer than the standard route 98% of the time. This result occurs regardless of which policy is used, although not favoring customers does benefit avoidance routing as shown. Further, even when 50% of the graph is removed, 95% of avoidance routes are no more than 10 hops longer than the original route, and 98% of routes are no more than 20 hops longer.

The second important result is to highlight how much worse neighbor-only avoidance routing performs. When only considering no-valley, neighbor-only has an increase of 5 hops or less for 98% of routes, rather than the 2 hops or less for SPF. Further, neighbor-only performs significantly worse when 50% of the graph is removed. Finally, as we've already seen previously, neighbor-only performs exceptionally poorly when favoring customers.

The CDFs for the geopolitical property distribution, removed from largest to smallest, is shown in Figures 5.27 and 5.28. As a reminder, since the US dominates the AS graph, the smallest removal we can do is 35%, rather than the 5% we did in the uniform removal. Further, we pick the removal percentage closest to 50% we can reach, which happens to be 52%. Figure 5.27 is the CDF when only no-valley is used, while Figure 5.28 is the CDF when favoring customers.

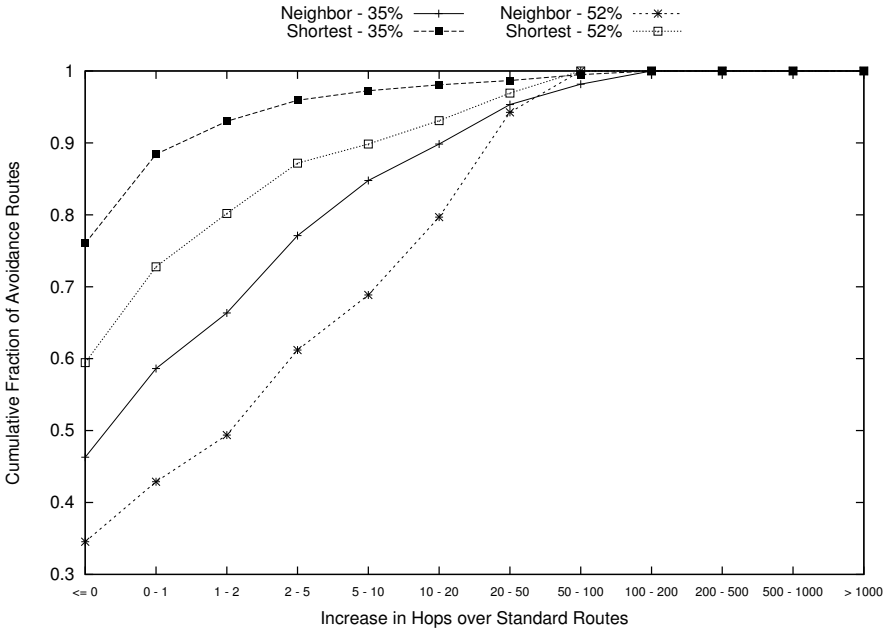

Figure 5.27: CDF of path length increase using geopolitical (largest) property distribution and "no-valley" policy

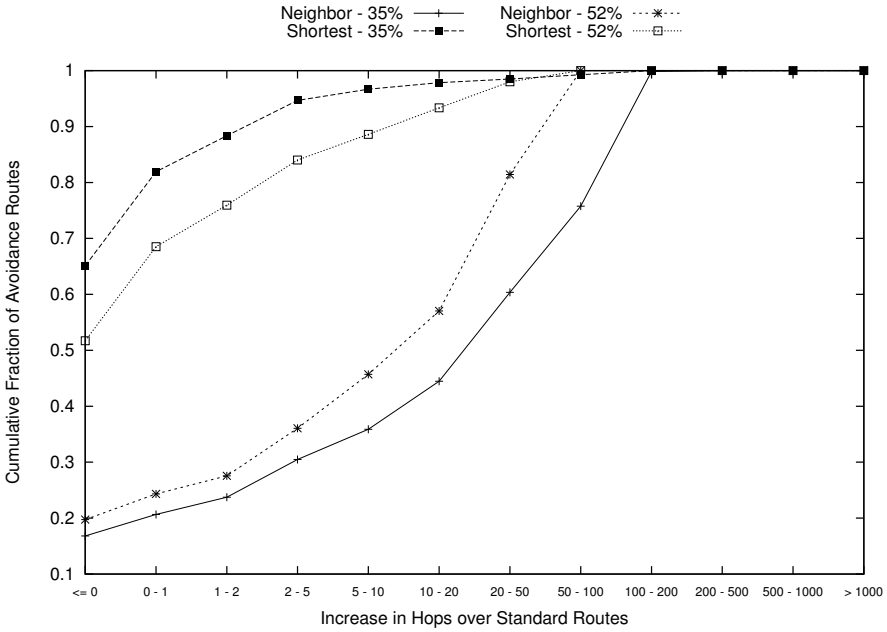

Figure 5.28: CDF of path length increase using geopolitical (largest) property distribution and favoring customers

These CDFs highlight and reinforce the points made by the uniform distribution. SPF does not perform nearly as well in the no-valley case, with 98% of paths having an increase of 50 or less hops; but it is important to remember that 35% of the graph, rather than 5% has been removed. The important result here, however, is that neighbor-only performs significantly worse than SPF. Further, when favoring customers, SPF performs very well, while the trend of neighbor-only performing poorly continues.

The overall takeaway from our evaluation of path lengths is that avoidance routing performs pretty well. The shortest path first heuristics is clearly the best in terms of avoidance path length. Further, we see that most avoidance routes do not experience a major increase in path length. Finally, we see that neighbor-only, the heuristic designed to forgo disseminating all the security properties in favor of privacy, performs worse than avoidance routing with the additional information. Why some of the effects we discovered occur will be discussed in Section 5.6. However, before discussing why certain things occur, we need to evaluate another portion of avoidance routing: extraneous nodes visited.

5.5 Extraneous Nodes Visited

The previous section led us to the conclusion that shortest-path first (SPF) is the best heuristic, outperforming all others in almost all cases. However, this is only the best heuristic for minimizing the length of the path found. It is possible that SPF visits a lot of extra nodes before finding the correct path. Another heuristic may find a longer avoidance route, but visit far fewer nodes. This is what we evaluate in this section.

In this evaluation, we will only be looking at the "no valley" policy and the "favor customers" policy. We will not be discussing the no policy case. As in the previous section, we discovered that no heuristic works particularly well in the no policy case. There are times when neighbor only is the best, and times when other heuristics are best. Why this occurs will be discussed in the discussion.

First we must define what extraneous nodes visited means. Extraneous nodes visited are the extra nodes visited during the search that do not actually contribute to the final discovered path. These are nodes on tested paths that lead to dead ends. To calculate these, we simply mark every node visited during a search in the simulator, and then subtract the number of nodes on the final path from this number.

Our first set of evaluations is for the "no valley" policy using the uniform and half-Gaussian distributions, shown in Figures 5.29 and 5.30 respectively. Here, we can see that, once again, SPF is the best heuristic, and neighbor-only is the worst. In fact, we see the same general breakdown from the previous section. Furthest is the worst of our standard heuristics, followed by nearest followed by least.

We can also see that there are a significant number of extra nodes visited by all heuristics. Even with only 5% of the graph removed, SPF still visit 5 extra nodes on average, and neighbor-only visits as many as 30 on average. As more of the graph is removed, we see SPF has a maximum average of around 40 additional nodes, and neighbor-only gets up towards 120 in the uniform distribution.

This is clearly not a great result. We would hope that this number remained close to 0 for our heuristics. That is, we hope the number of bad choices made would remain small. Unfortunately, that appears not to be the case. Figure 5.31 shows the same trend for the geopolitical property distribution. This is the distribution removed from largest to smallest. In fact, we see the effect exacerbated, with SPF visiting nearly 50 extra nodes when the network is 60% removed.

The geopolitical removal from smallest to largest is not shown as the number of extra nodes visited is small. This is due to the exact same reason the overall growth in path lengths was small for this distribution in the previous section. The majority of the Internet is dominated by the largest countries, so even with most of the graph removed, it is easily possible to find a path completely contained by these large countries.

Why do these graphs peak around 40 – 60% rather than at 75%? This is due to the

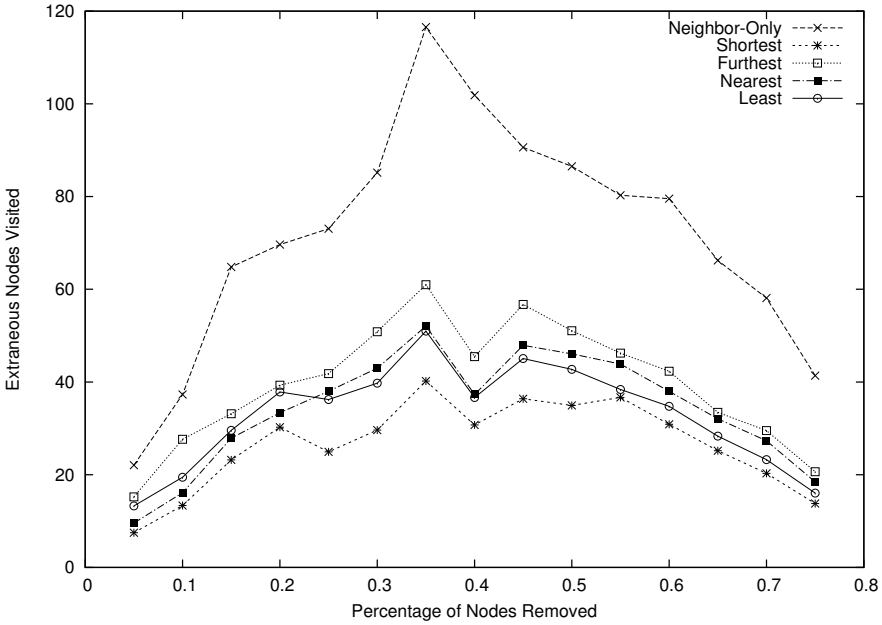

Figure 5.29: Extraneous nodes visited for the uniform property distribution and the "no valley" policy

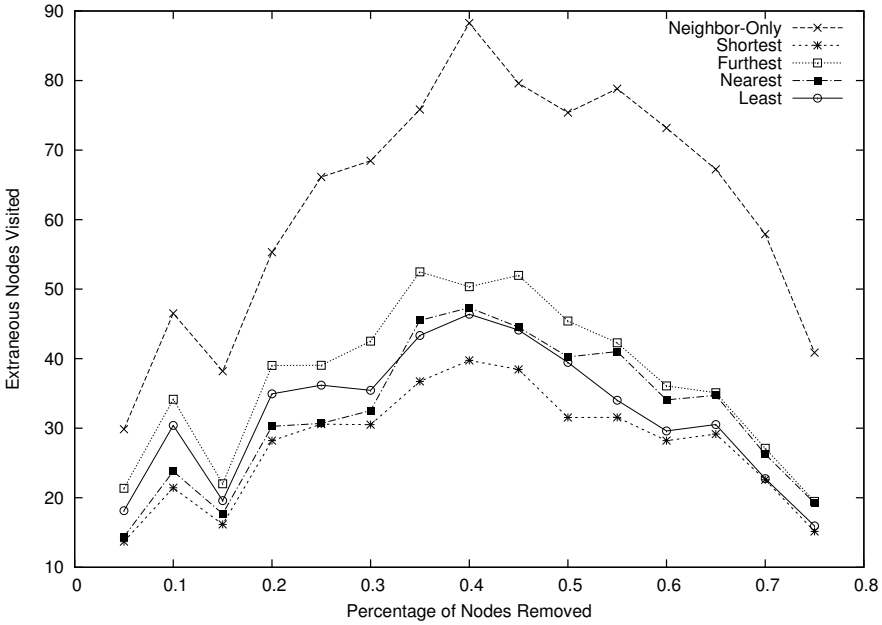

Figure 5.30: Extraneous nodes visited for the half-Gaussian property distribution and the "no valley" policy

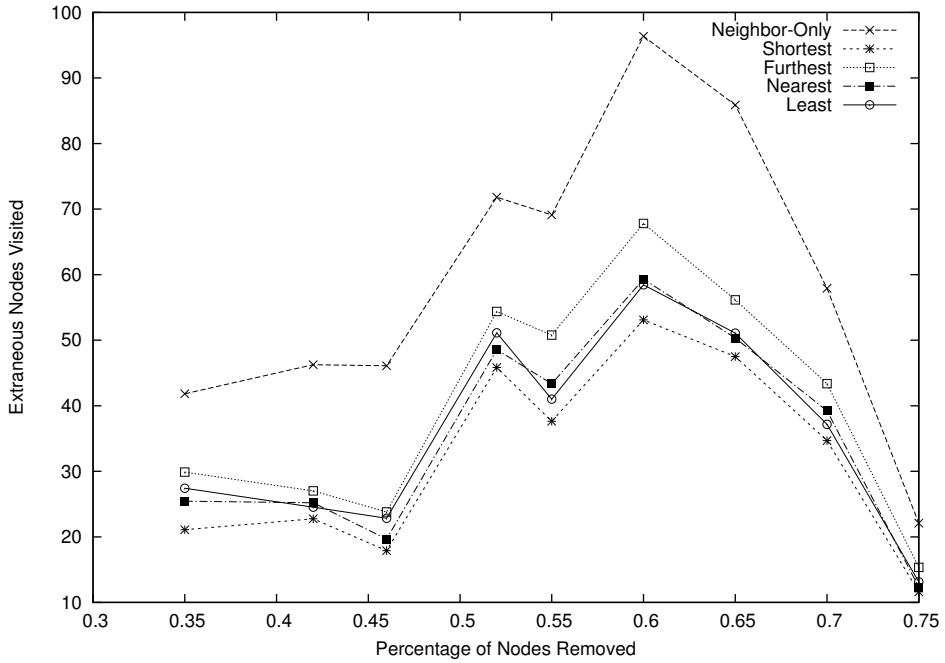

Figure 5.31: Extraneous nodes visited for the geopolitical property distribution and the "no valley" policy

fact that overall path lengths decrease after we reach a tipping point of around 50%. That is, when 50% or more of the graph is removed, it is unlikely to traverse a large number of nodes before running into a dead end. This is the same reason why we saw the same general effect in the previous section.

We can see the effect of favoring customers in Figures 5.32, 5.33 and 5.34 for uniform, half-Gaussian and geopolitical property distributions. First, we see the same general trend we saw before in the discussion of path lengths, neighbor-only performs exceptionally poorly. We also see the same basic result we saw when not favoring customers, the overall extra nodes visited is still rather high. Again, SPF will visit around 50 additional nodes, on average, in the worst case.

When standard deviation is included, we see that the range of possible extra nodes visited is very large. Figure 5.35 shows that avoidance routing may visit a large number of extra nodes. This figure displays the results for the uniform property distribution when

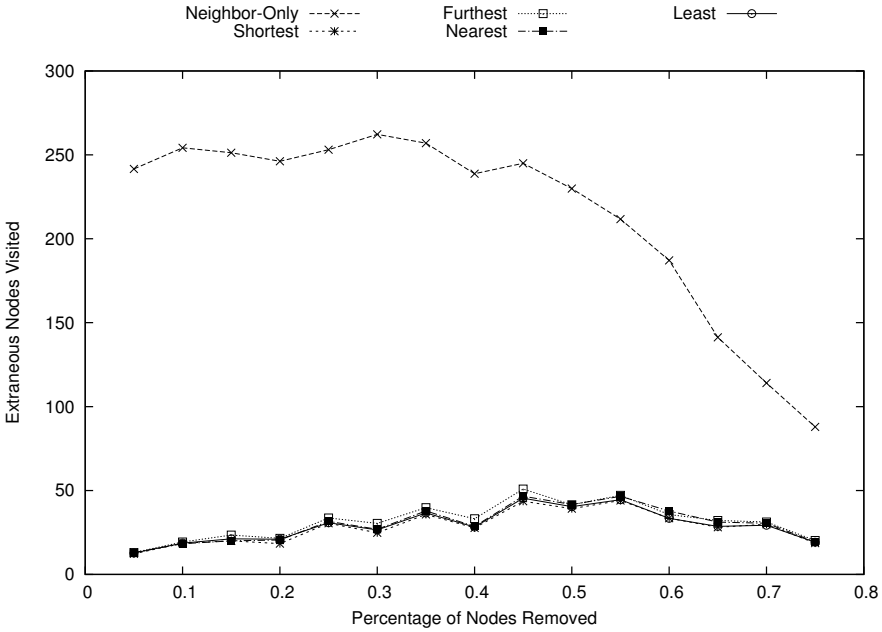

Figure 5.32: Extraneous nodes visited for the uniform property distribution and favoring customers

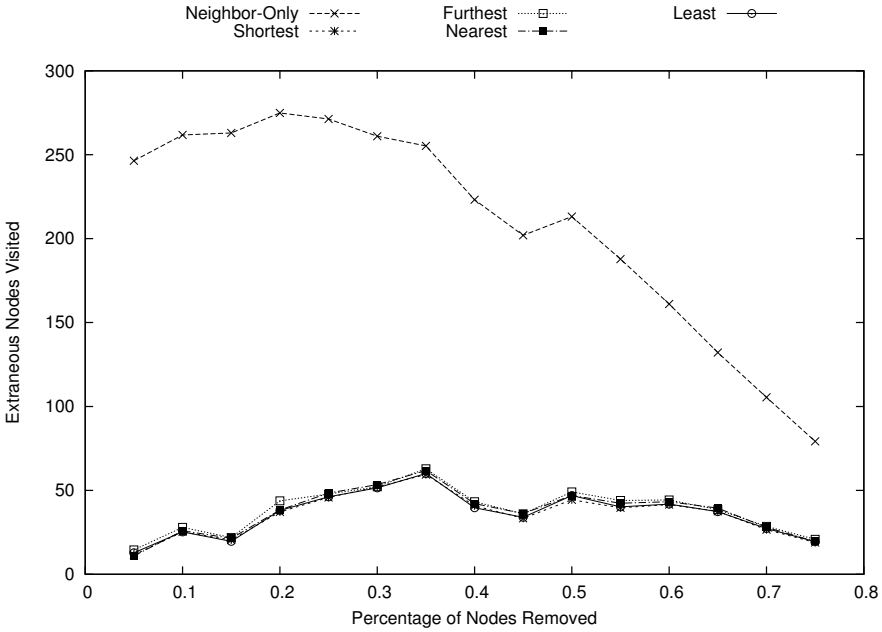

Figure 5.33: Extraneous nodes visited for the half-Gaussian property distribution and favoring customers

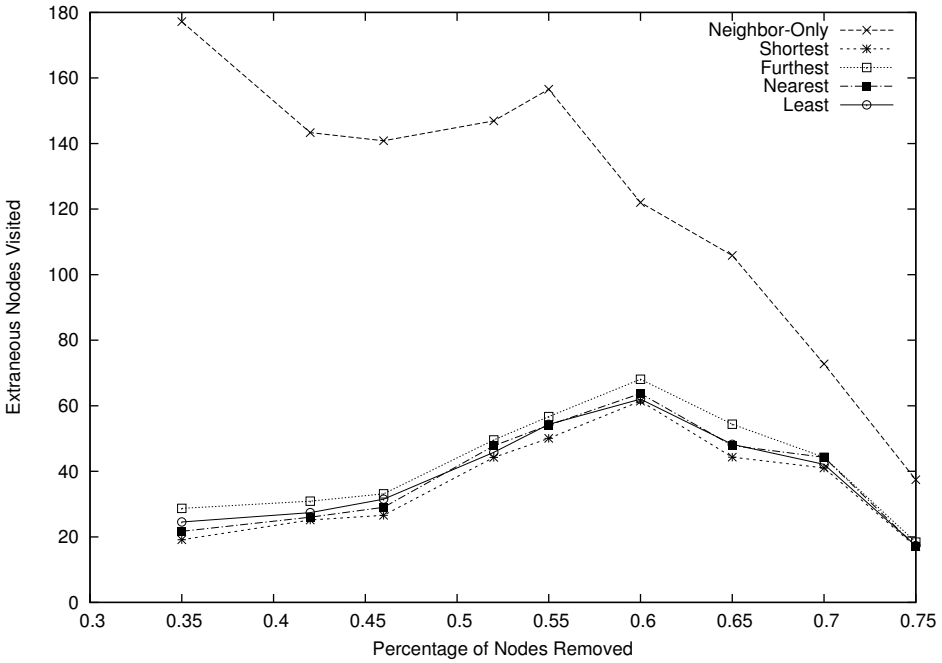

Figure 5.34: Extraneous nodes visited for the geopolitical property distribution and favoring customers

not favoring customers. Three standard deviations are shown as well, giving us a 99.7% confidence. Only the shortest-path first heuristic is shown. The results shown are representative of the other heuristics, property distributions and the favor customer policy. Clearly, visiting up to 600 extra nodes is a waste of resources.

The next obvious question is what is the maximum number of extra nodes visited? This information is shown in Table 5.2. As shown by this table, the answer is not very good. There are cases where twenty-five hundred nodes are visited. While this represents less than 10% of the AS graph, it still is far more nodes than are required. In fact, in the case where 2518 extra nodes were visited, the final discovered path was only 8 hops long.

While it is possible that is the result of an error in our simulator, this is probably not the case. Recall that in a depth-first search, if you make a bad decision, you are committed to it until you exhaust that entire portion of the graph. It is much more likely that a specific bad choice is leading to the significant increase in nodes visited. A poor choice made at

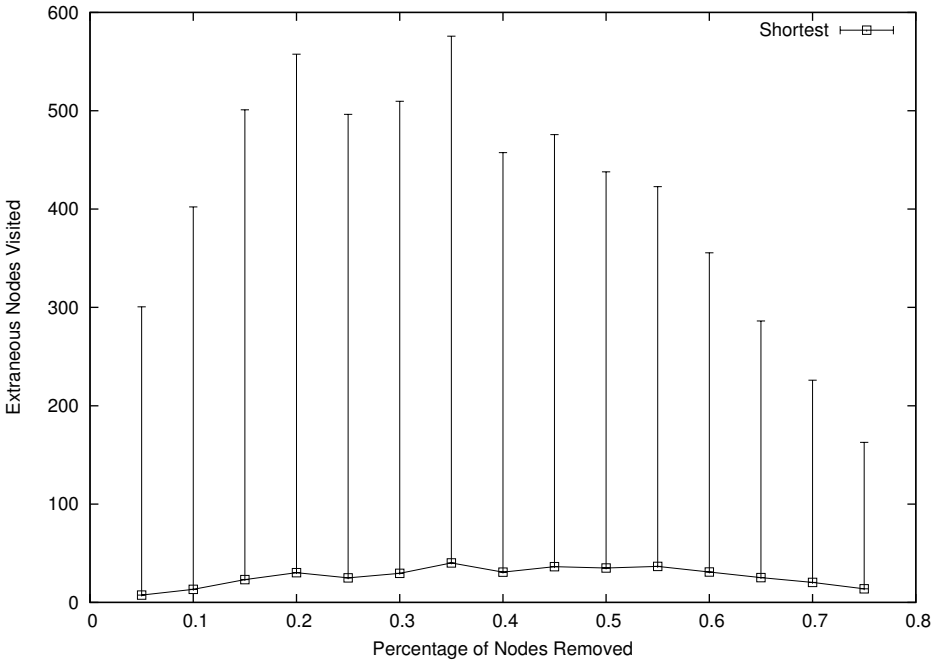

Figure 5.35: Extraneous nodes visited with policy for the uniform property distribution

the first or second hop may cause the search to visit an entire portion of the graph that cannot reach the destination. When this entire portion is exhausted, the node that made a bad choice can finally make the correct choice. This is a known problem with DFS.

Before panicking that avoidance routing makes bad choices, we should understand the frequency by which these bad choices are made. Just like path length, it is possible that bad choices are infrequent. If one were just looking at standard deviation and average, those results might suggest that bad choices are quite frequent.

To understand the frequency of bad choices, we will again be using the cumulative distribution function. Figures 5.36 and 5.37 show the CDF of extraneous nodes visited for the uniform property distribution, with the former displaying the "no-valley" policy, and the latter favoring customers.

These results show us that it isn't quite as bad as we thought. With only 5% of the graph removed, shortest-path first almost never visits any extraneous nodes. Only 1% of routes found will visit extraneous nodes, although these do tend to visit a large number of

Table 5.2: Maximum extraneous nodes visited for different property distributions (Percent of graph removed)

Policy	Heuristic	Uniform	Half-Gaussian	Geo (Largest)	Geo (Smallest)
No Valley	Neighbor	2182 (10%)	1879 (20%)	1274 (35%)	1767 (25%)
	SPF	2014 (5%)	2013 (10%)	1396 (35%)	1766 (25%)
Favor Customers	Neighbor	2399 (5%)	1939 (10%)	1144 (35%)	1829 (35%)
	SPF	2518 (5%)	1823 (5%)	1064 (35%)	1311 (5%)

extraneous nodes. With 50% of the graph removed, SPF does not perform as well, although only 10% of routes will visit more than 10 extraneous nodes.

These results further reinforce the fact that neighbor-only is not a practical avoidance routing variant. In either policy version, neighbor-only performs worse than SPF. Further, with 50% of the graph removed, it performs significantly worse, with less than 60% of paths found visiting no extraneous nodes. By comparison, SPF visits zero extraneous nodes almost 90% of the time. Finally, neighbor-only is significantly hampered by favoring customers, as we had seen throughout all our results.

Figures 5.38 and 5.39 show the CDFs of the geopolitical property distribution. The same basic trends hold. SPF out performs neighbor-only by a significant margin. SPF does not perform as well as it had in the uniform property distribution; however, recall that this distribution begins with 35% of the graph removed, rather than 5%. It is worth noting that more than 90% of searches visit zero extraneous nodes when the US has been removed from the Internet graph. This is true regardless of which policy is currently in use.

We would not call having 5% of searches visiting at least 50 extraneous nodes necessarily a positive result. While these searches are visiting less than 0.2% of the AS graph, they are still using resources on these nodes. Further, the occasional search that visits 2,000 extraneous nodes is causing a large amount of resources to be used on nodes that cannot reach the destination. Mechanisms to mitigate this, as well as other interesting results

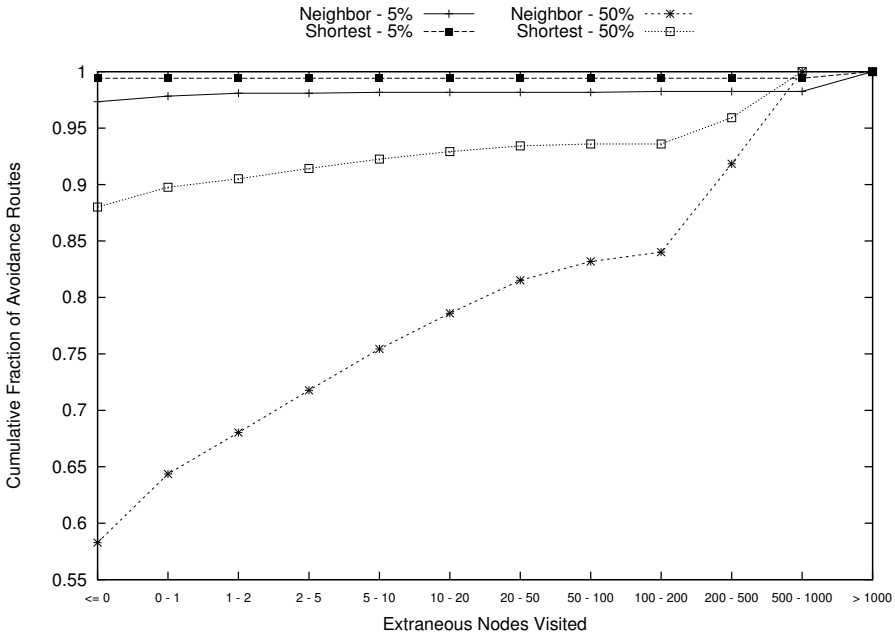

Figure 5.36: CDF of extraneous nodes visited using uniform property distribution and "no-valley" policy

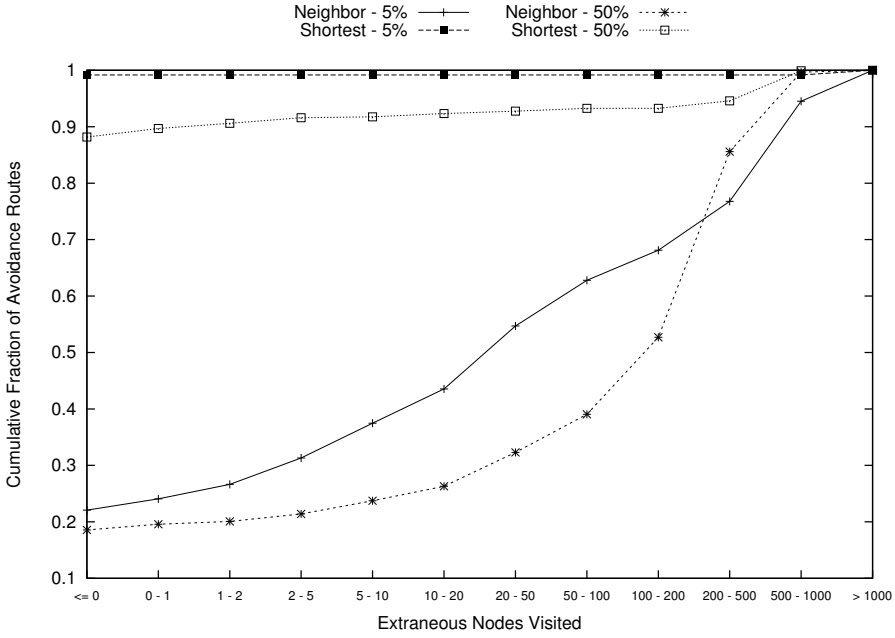

Figure 5.37: CDF of extraneous nodes visited using uniform property distribution and favoring customers

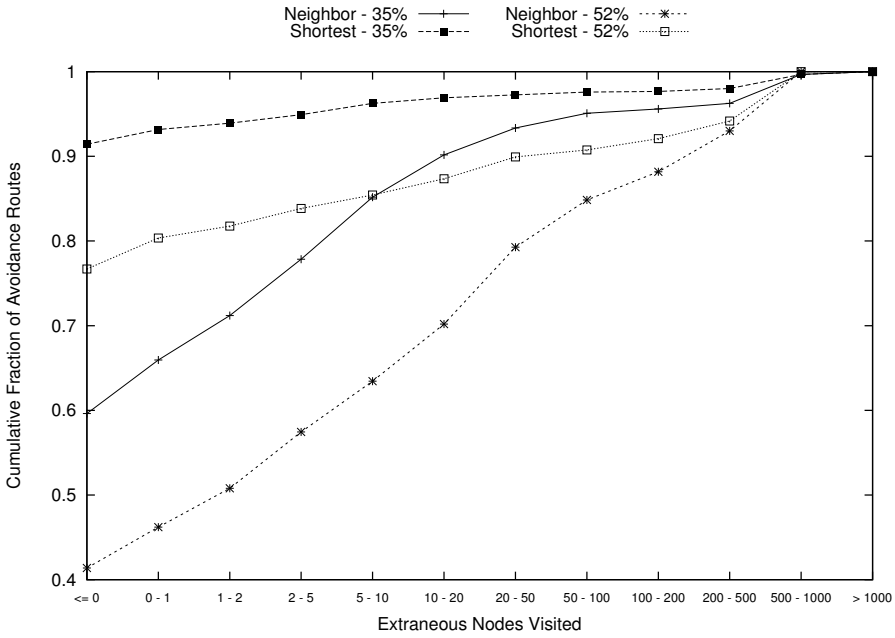

Figure 5.38: CDF of extraneous nodes visited using geopolitical property distribution and "no-valley" policy

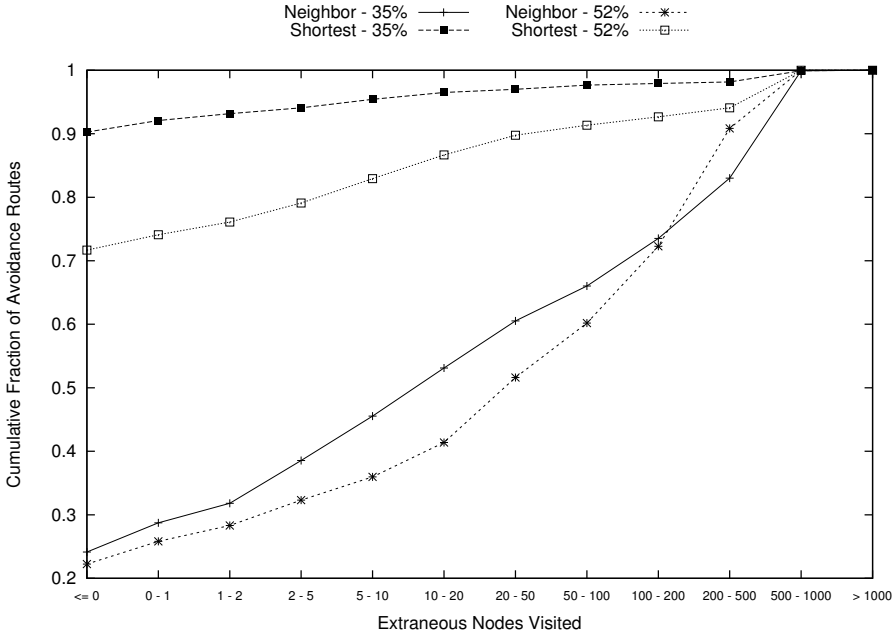

Figure 5.39: CDF of extraneous nodes visited using geopolitical property distribution and favoring customers

about avoidance routing, will be discussed in the next section.

5.6 Discussion

We've learned several things from our results. The first thing we've learned is that shortest-path first is the best heuristic, with neighbor-only being the worst. Why does SPF perform so well and neighbor-only perform so poorly? Let us focus on why neighbor-only performs so poorly. The simple answer is that neighbor-only is performing a "blind" search. It does not have the information available to the rest of our heuristics in terms of which paths are valid or where collisions may occur.

How important is this information? Clearly, it is giving an advantage to the other heuristics. In fact, the advantage is rather significant, in terms of quickly finding a path and not visiting extraneous nodes. Figure 5.40 displays the percentage of avoidance routes where one of the first **two** hops had a completely valid advertisement. This represents the uniform property distribution and policy is in use. This graph shows that a majority of the time, one of the first two hops already knows a valid path. In fact, even with 65% of the graph removed, more than 75% of valid avoidance routes are known within the first two hops.

The importance of this information cannot be understated. This means that 75% of the time, at most **one** node will enter the search phase. Only one node will be required to set up search structures and use resources. The remaining nodes will search their advertisements and discover a completely valid path immediately. Thus, they will simply forward the request based on the valid advertisement and set up the appropriate caches. No search resources will be used for these requests.

Neighbor-only is not afforded this luxury. It cannot abort the search by discovering a completely valid path as it does not have the requisite information. Further, every node on the path will be required to enter the search phase, unlike with the other heuristics. In

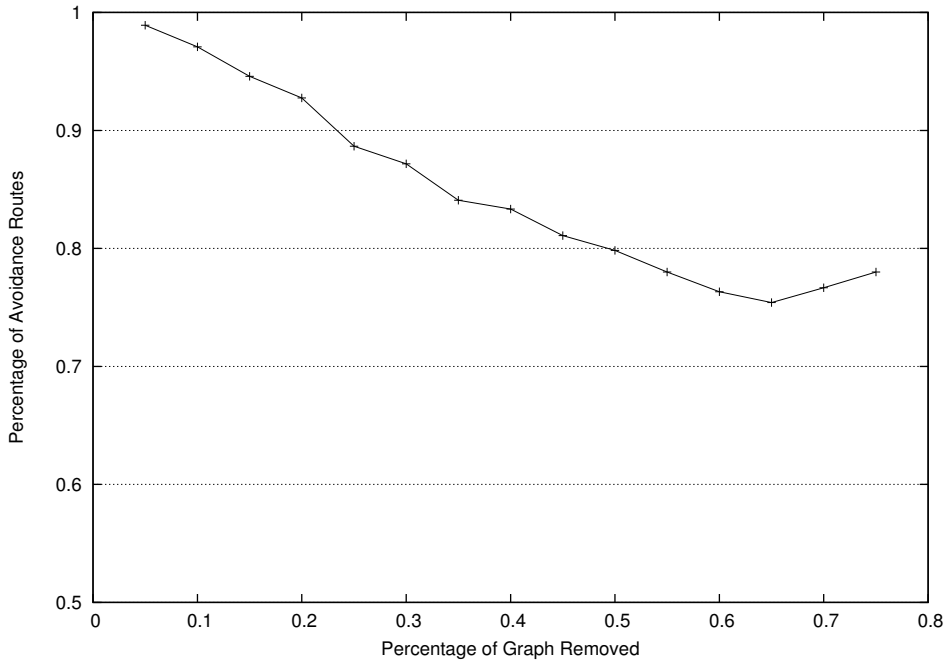

Figure 5.40: Percentage of avoidance routes discovered by a valid advertisement within the first two hops, with the uniform property distribution

fact, neighbor-only will have to blindly find its way to the destination, possibly going down many dead-ends, when our other heuristics will simply skip the search.

The effect is not quite so pronounced when using the geopolitical distribution, and removing properties from largest to smallest. This result is to be expected. Remember that the largest countries dominate the Internet graph, as well as the Internet core. Since the majority of advertisements go through the core [MK11], it is reasonable to conclude that if the core is invalid, these advertisements will not be valid.

Figure 5.41 shows the percentage of routes that are discovered via a valid advertisement within the first two hops, for the geopolitical distribution. As stated, the results are not as useful. With only 52% of the graph removed, less than 65% of routes can be found within the first two hops. In fact, with 70% of the graph removed, less than 50% of routes will be valid within the first two hops. As explained, this is not surprising. Removing the Internet core will undoubtedly reduce the chance of finding a valid path quickly. Anyone excluding

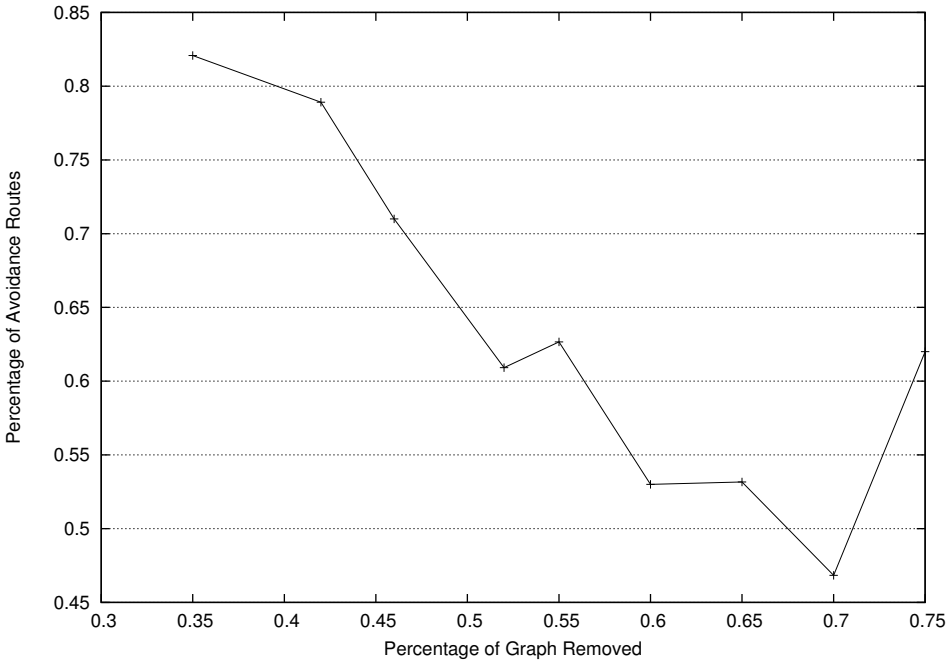

Figure 5.41: Percentage of avoidance routes discovered by a valid advertisement within the first two hops, with the geopolitical property distribution

that much of the Internet should anticipate a longer path setup.

The additional information in the security vector clearly explains why SPF performs better than neighbor-only. Further, it highlights that not only will SPF find shorter paths, but also find them faster and with less resources used. However, why does SPF perform better than the other heuristics? The main reason is that a heuristic that favors shorter paths will tend to find shorter paths. But this doesn't necessarily explain why SPF doesn't visit more nodes than any other heuristics.

The answer behind this is due to the connectivity of the Internet. A large majority of paths transit the core of the Internet. Any path selection will tend to reach the core before heading out to the edge. A heuristic that favors shorter paths, once in the core, will tend to find the shortest path to a destination. The other heuristics may tend to find longer paths and visit far more nodes while doing so.

Shortest-path first was not *always* the best heuristic. In some cases, other heuristics

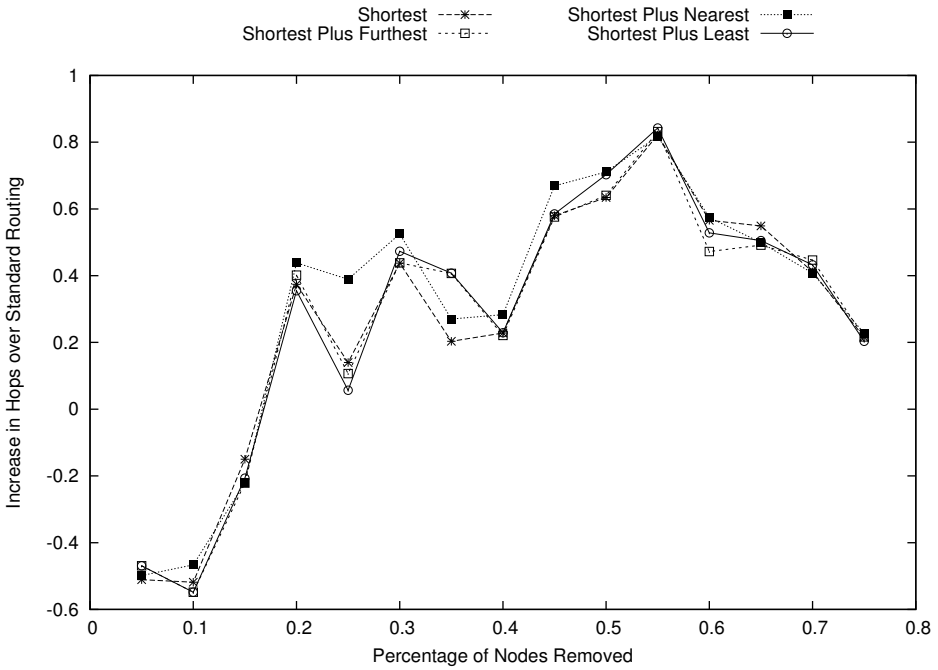

Figure 5.42: Increase in Path Length when chaining heuristics

would perform better. This was especially true when most of the graph had been removed and the advantage of using shorter advertised paths was lost. However, no heuristic was generally better than SPF. Further, it was not obvious what network conditions lead another of the heuristics to be better for any one set of constraints. Without being able to determine this quickly from the request and destination, a router must always use the heuristic that is better on average. It may possible in the future to leverage the security properties in ways we haven't thought of to determine routes more quickly and more accurately.

Another question is whether "chaining" the heuristics together will provide a benefit? This means first sorting by SPF and then sorting by another heuristic. Figure 5.42 shows an example of the effect on path length when chaining heuristics. This example uses the uniform property distribution and the no-valley policy. Although it is hard to see, there are times when each heuristic outperforms the others. However, these differences are very subtle. In general, chaining heuristics performs no better than simply using SPF. These

results are similar in the other distributions as well.

We now know why SPF is the best heuristic, but why does avoidance routing perform better when policy is introduced? Further, why are some avoidance routes **shorter** than the standard BGP route? First, in general, BGP routes when policy is used are longer than those when policy isn't used. The constraints introduced by policy reduce the set of possible routes between any two nodes.

There is a simple proof to show that routes in BGP with policy will be tend to be longer than without policy. First, without policy, every route will be the shortest possible route from source to destination. Now, remove some of these routes from the selection and pick new routes. Since the original route was the shortest route, it is impossible for the routes to get any shorter. Thus, the routes must be as long, or longer than the original routes.

Thus, avoidance routing performs better partly because BGP performs worse. The relative and absolute increase in path length will decrease when the base lengths increase. This is only part of the reason. The other reason has to do with the structure of the Internet and the nature of shortest-path routing. Shortest-path routing will always use the shortest-path, regardless of if it goes through the core or not. Further, the routes from many nodes to the same destination may tend to have significant overlap, since each route attempts to be the shortest. The combined effect is that each node may have less information about alternate routes and may be less likely to go into the highly connected core, where many alternate routes exist.

This explains why avoidance routing performs better when given policy, but why does it sometimes perform even better than standard routing? Recall that when using policy, nodes prefer to use customer routes over provider routes. Consider the example shown in Figure 5.43. In this case, S wants to find an avoidance route to D, avoiding gray. The standard BGP route would go "down", through customer C, rather than take the shorter provider route through P. However, the standard route is invalid. But node S has a completely valid route that it is aware of, through P. Thus, avoidance routing will discover a "better" route

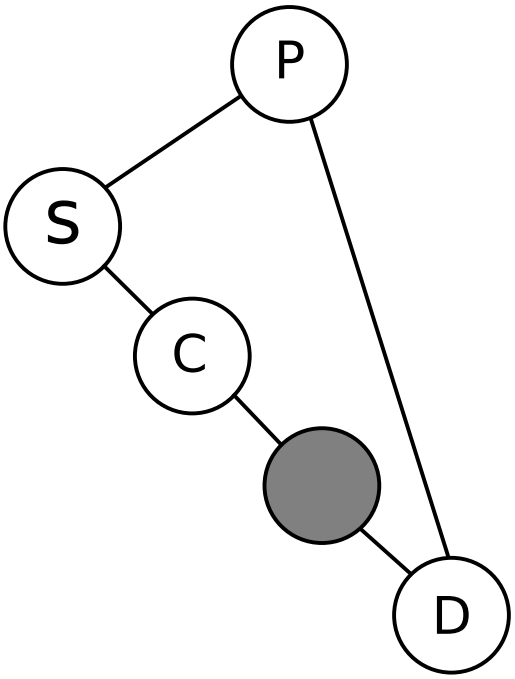

Figure 5.43: Topology when avoidance routing is better than BGP

to the destination.

Its possible that an unknown customer route reaches the destination. Avoidance routing will never discover this route if a known valid path exists. This is the behavior we expect from avoidance routing. Avoidance routing assumes that using a fully known route is better than the existence of a "better" unknown route. We believe this assumption is valid as using a known path ends up using a known amount of resources. Trying to find a "better" unknown path may end up using far more resources.

One point should be clarified here — avoidance routing will only use paths that do not violate policy. While our experiments show that avoidance routing can find "better" paths to a destination, it is possible that these paths violate some other policy besides the ones tested here. In these cases, the avoidance routing search will determine that these paths are not usable and will not try them. We assume that the paths used by avoidance routing are either advertised valid paths or unadvertised valid paths as discussed in Section 4.2.

So avoidance routing performs better than the neighbor-only variant, as well as performing better when policy is included. There is still an open question about the neighbor-only variant — why does it perform so poorly when favoring customers? The answer to this question includes several factors, some of which we've already introduced.

The first factor is that whenever a customer route is used, the paths is moving away from the core. The second is that the further you get away from the core, the more sparse the Internet graph becomes. Thus, if one is favoring customers, they are moving away from the core and have less options to exercise as they get further from the core. Neighbor-only suffers from this problem. They pick routes blindly that move further from the core. Since DFS penalizes bad choices, neighbor-only will struggle downward happening upon the one, long route, that reaches the destination.

The question then becomes, why doesn't avoidance routing suffer in general? This is, once again, due to the extra information that standard avoidance routing has. Both neighbor-only and standard avoidance routing will, at some point, find themselves in the Internet core. Neighbor-only will favor a customer route and, thus, make a bad decision. Standard avoidance routing will discover a valid path through a peer and use it immediately, rather than suffering as neighbor-only does. This is why neighbor-only performs much worse when favoring customers.

Another interesting question is, why does neighbor-only perform well at all? We've gone through a significant study to show that neighbor-only performs poorly when compared to our other heuristics. But one might ask, why does it not perform worse? This is due to the fact that neighbor only is using the standard BGP routing mechanism. It will always pick the path that BGP picked first. This leads us to a question we asked in the beginning of this chapter, how often does the standard BGP route satisfy the avoidance request?

Figures 5.44 and 5.45 display the percentage of time the avoidance path is the standard path. Figure 5.44 shows this for all three policies using the uniform property distribution, while Figure 5.45 uses the geopolitical distribution. The error bars shown are three standard

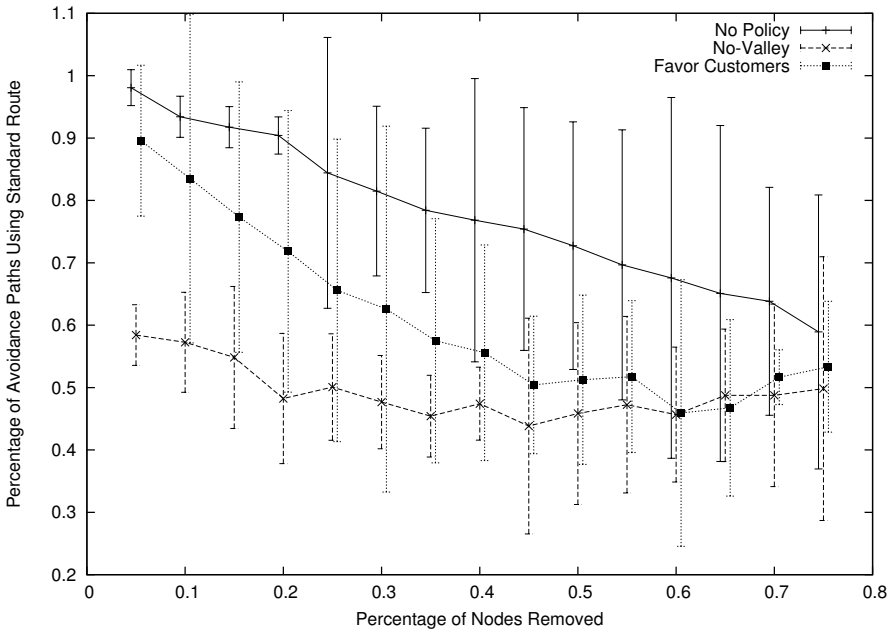

Figure 5.44: Percentage of routes that are the same as the standard route with the uniform property distribution

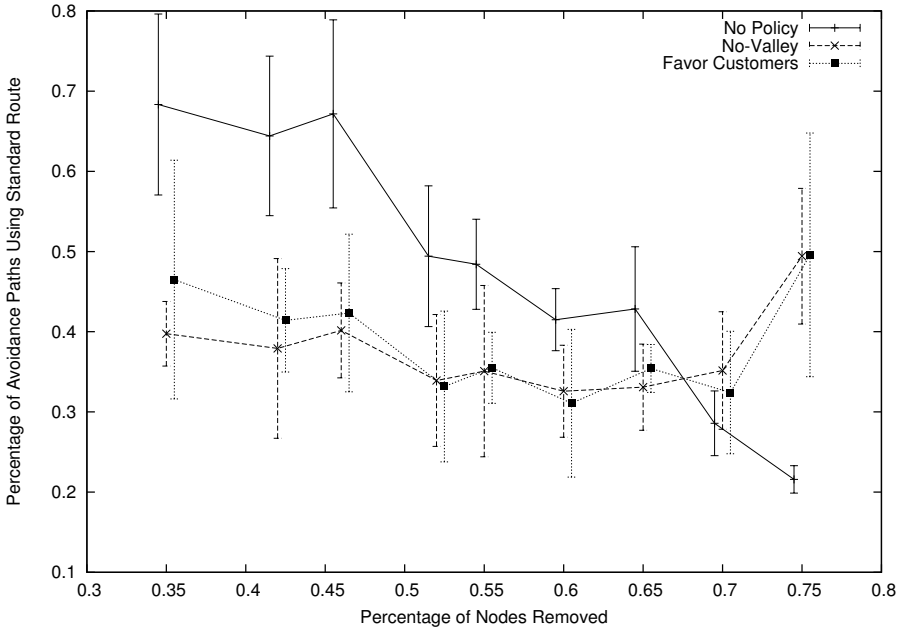

Figure 5.45: Percentage of routes that are the same as the standard route with the geopolitical property distribution

deviations. In the case of no policy, we see a steady decline in path sharing as more of the graph is removed. This is due to the fact the no policy case results in shortest paths, reducing the likelihood of having knowledge of additional paths.

For uniform properties, if we favor customers, we also see a steady decrease in sharing until most of the graph is removed, at which point we see an increase in sharing. Once most of the graph is removed, the remaining paths tend to be short paths between source and destination, and may have already been the path shared in the beginning. We see a similar trend with the geopolitical removal strategy, although sharing is greatly reduced. Once again, this is caused by the tendency for this removal strategy to remove the Internet core.

Finally, when considering the "no-valley" policy, we see that sharing tends to begin low and remain low. This is caused by the fact that the avoidance route may preferentially go through the core. In contrast, the standard BGP route will preferentially go through customers. As we discussed previously, this may result in a shorter avoidance path than the standard path.

So why does neighbor-only avoidance routing perform well at all? These results show us that in all of our experiments, at least 20% of avoidance routes are the standard route. In fact, the majority of our experiments show us that at least 35% of avoidance routes are the standard route. In the case of any avoidance routing, including neighbor-only, the avoidance route to the destination is already known by the source in these examples. Thus, 35% of the time, neighbor-only already gets it right. It is the remaining 65% of the time that neighbor-only will perform poorly.

The last point that needs to be addressed is the information we learned in the previous section. We learned that a significant number of additional nodes may be visited during a search. Clearly, any extraneous node that is visited during a search is wasting its resources. By the nature of using a on-demand search, it is unreasonable to assume that no additional nodes would be visited. One may even argue that visiting 50 additional nodes is acceptable.

This is, after all, a small portion of the Internet graph.

Still, visiting up to 2000 nodes is not acceptable. Clearly, a smart attacker could determine a path that utilized the most nodes. While our results show that this does not occur very often, it is possible to be minimized. One could set a path length limit. As we know, most paths tend to be short. If we set a path length limit, it would reduce the number of nodes visited as well as increase the chance of finding a short path.

Unfortunately, it may also reduce our path independence. Recall that path independence says that decisions made at a node are regardless of how one reaches that node. If path length becomes a constraint, then if one reaches a node with a shorter path length, the node will have to conduct its search again. How this exactly plays out is subject of future research.

A better way to address this problem is to first understand what causes it. There are two main causes. The first cause is that DFS penalizes bad decisions. The second cause is that there may only be one node that actually has a path to the destination in a region and the remaining nodes all use that node. If this node is invalid, every path through this "region" will fail as they attempt to use this node. Thus, a big mitigation mechanism would be to check to see the similarity of advertised paths, and try to use dissimilar paths after failing.

This mitigation strategy may be accomplished by creating a partial map of the topology based on the information in the BGP advertisements. This may allow for better path selection by determining where collisions occur and if advertisements share similar paths or have the same collision. If one of these paths fail, it might be better to try another unrelated path next rather than the similar one. This will also be discussed more detail in Chapter 7.

A final mitigation strategy would be to conduct parallel searches. Thus, in theory, the search that visits the least nodes would conclude faster. How parallel searches could work, and there drawbacks, will also be discussed in Chapter 7. While we've determined how well

avoidance routing performs when choosing routes, we still need to determine the overhead the search causes. This will be discussed in the next section.

5.7 Search Overhead

There are several factors which determine the overhead involved in a search. The first is the size of the search entry, the data stored during a search. The size of the search entry is determined by two factors. Everything has a fixed size except the list of interfaces to try. The source IP, destination IP, incoming interface and criteria hash combine to a total of 16 bytes. The size of the interface list depends on the total number of interfaces the router has. The Juniper TX Matrix Plus has approximately 1024 10G ports [Jun12]. This would require 10 bits per interface for a total of 1280 bytes if all interfaces are in the list. Thus, 1 MB of RAM would support 862 individual searches. This clearly scales linearly, so 100 MB of RAM could support 80,000 searches.

The overhead of the search is also determined by the CPU spent in sorting the interfaces based on the advertisements given. A standard sort of the advertisements will result in O(N log N) overhead, where N is the number of known advertisements. Since each neighbor only advertises a single path to any destination, N is bounded by the number of neighbors. Assuming that all neighbors are valid, and a node can have 1024 neighbors, than this search is bounded by 10,240 comparisons, which is relatively small.

Further, as shown previously, a large number of nodes will have a valid advertised path to the destination. If the sort uses quick-sort, than the first pass of the sort will discover the valid path. Thus, any node that has a valid path will have to check at most O(N) nodes. In the 1024 neighbors case, this is at most 1024 entries to check.

The fact that some nodes will have a valid advertisement means that these nodes will not have to enter the search phase. As we showed before, as many as 90% of paths would only require one node to enter the search phase. Even for paths that do need multiple

nodes to enter the search, some set of nodes near the destination will not need to enter the search phase. For all of these cases, no resources will need to be spent for the search. The memory overhead for these nodes during the routing search is effectively zero.

There is one final point that should be discussed. If shortest-path first is the best heuristic, does that mean there is too much information about security properties in the advertisements? Recall that SPF sorts routes if either they are completely valid, or if they are the shortest. The length of the route is held in the path-vector, which is already included in BGP. To determine if a route is completely valid, avoidance routing checks the entries in the security-vector, which correspond one to one with the ASes in the path-vector.

But some of this information is redundant. The security-vector was used to allow ASes to use other heuristics, like furthest collision. SPF does not make use of that information. Instead of using a security-vector, we could simply have a list of properties that exist on the path. Thus, if two ASes are both in the US, the US would only appear once.

There are several reasons to keep the security-vector. The first is that a new heuristic that does use this information may come into use later. Having to modify avoidance routing to use a security-vector after having removed it may require significant changes to a deployed protocol. Second, if we use this security list rather than the vector, each AS on the path must search the list to see what is listed, and then add whatever hasn't been listed. Since the list is in sorted order, it would require having to modify in place or duplicate the list in order to make changes. The process of simply adding the properties to the end of the vector is far simpler. The third and final reason to keep the security-vector is perhaps the most important. We use the information contained in the security-vector as a mechanism to validate information and detect liars. This process is described in Chapter 6.

5.8 Evaluation of Forwarding

The final piece of the puzzle is to determine how well avoidance routing actually forwards packets. Avoidance routing should not penalize any packet that is not using avoidance routing. That is, the mechanism used to determine if a packet is an avoidance routing packet should not burden standard packets. Further, packets that are being routed via avoidance routing should not pay a heavy penalty for doing so.

To evaluate this, we ran two different experiments. Both experiments use our avoidance routing implementation in the Click Modular Router. In these experiments, a client attempts to fetch a five megabyte file from a server. The file is fetched 1000 times and there is no delay between the end of one transfer and the beginning of the next. Further, the file is not cached in any manner so the entire file is transferred in each run. Finally, the length of the path is varied from 3 hops to 19 hops.

We measure two standard quantities to determine the affect on packet forwarding – connection duration and time to first byte. The first metric informs us if there is any penalty paid in aggregate through a large number of data packets. The second metric determines if any penalty is paid up front which may be mitigated throughout a large data stream. We also measured throughput, but decided not to show this as the throughput in all our tests was identical.

5.8.1 Non-avoidance Packets

In order to test the effect on non-avoidance packets, we compare our avoidance router to one of the standard Click routers. In this experiment, we used Click's linear router, which is their simplest router. While this is not a router type optimized for speed, we felt that the simple nature of the topology (only 20 IP addresses) meant that a linear look-up would perform no differently than a more complex look-up. The avoidance router uses a hash table to look up and route standard packets.

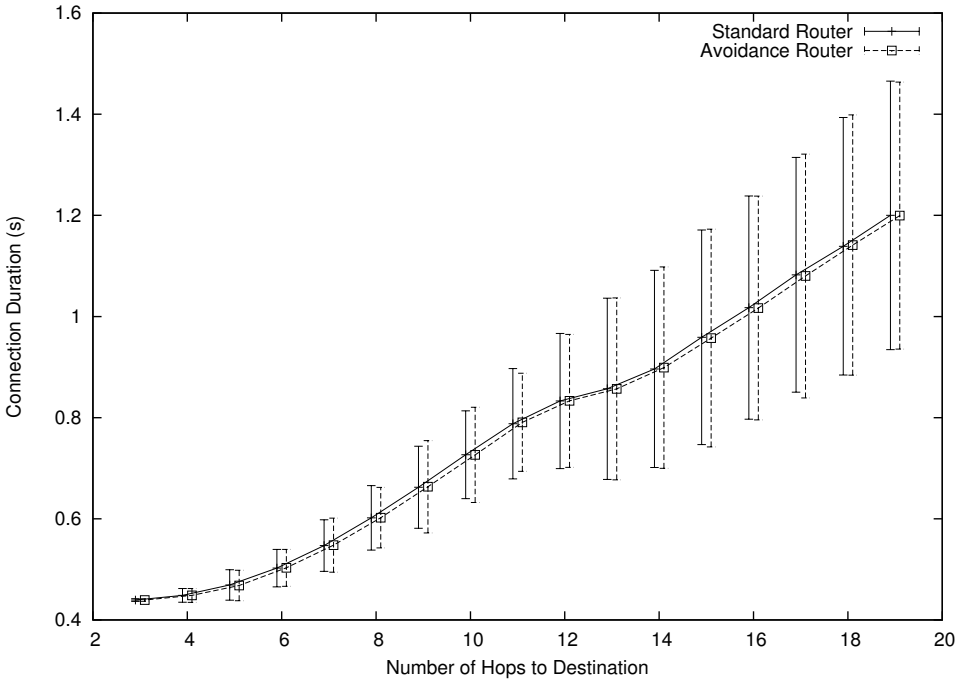

Figure 5.46: Connection duration for non-avoidance routed packets

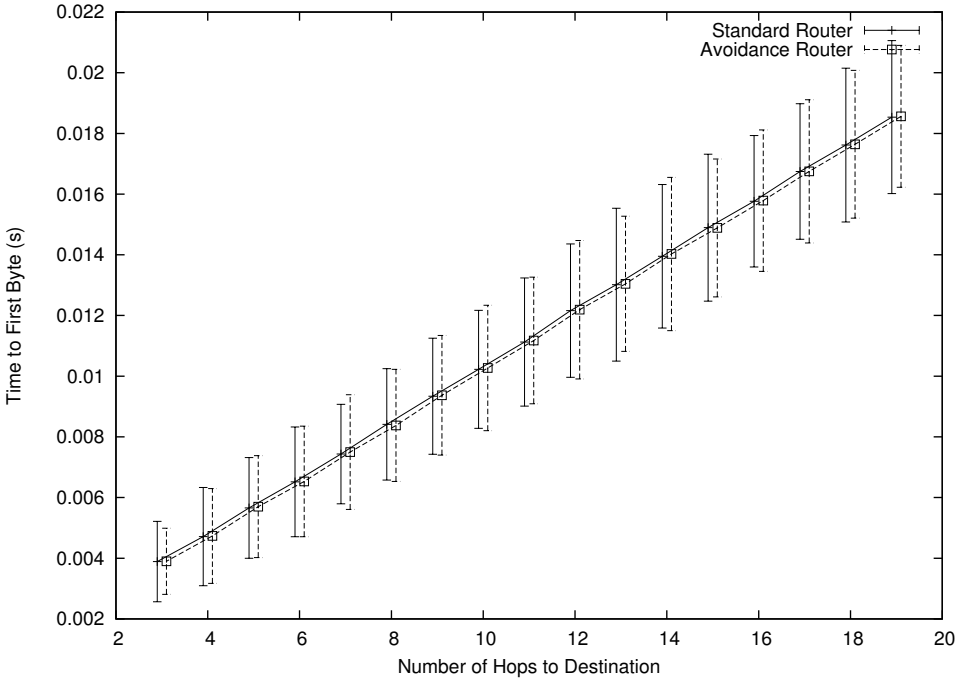

Figure 5.47: Time to first byte for non-avoidance routed packets

Figure 5.46 shows the connection duration for non-avoidance routed packets for the standard router and the avoidance router. Note that the data points are slightly offset to make the data more readable. This creates the illusion that the avoidance router performs better than the standard router. To be clear, this result shows that there is no difference between the avoidance router and a standard router in terms of standard packets. Our results for time to first byte, shown in Figure 5.47, reinforce this result.

This result is not surprising. The primary difference between the avoidance router and the standard router is that the avoidance router checks to see if the packet is an avoidance request. It does this by checking the protocol field in the header, which a standard router would not. A single additional comparison will not add much additional overhead. While the avoidance router is also looking for packets with the avoidance routing IP option, a standard router must also check for IP options. Thus, both routers pay this penalty.

5.8.2 Avoidance Packets

The final question is how does avoidance routing perform when moving avoidance routed packets? To understand this, we once again compare our avoidance router to Click's linear router. This time, avoidance routing is used to move packets to the destination on our avoidance router. It is critical to mention that the path is constructed prior to running the experiment. The goal of this experiment is to understand that penalty of our forwarding mechanism, not the path discovery mechanism.

The results of this experiment are shown in Figures 5.48 and 5.49. Again, the data points are offset for readability. These results show that there exists a negligible penalty for using avoidance routing. We see an increase in connection duration of only 10 ms over 20 hops. The increase in time to first byte is less than one millisecond. These increases are small enough to be considered insignificant. There may be an application that cannot absorb this small increase. In those cases, they will not be able to use avoidance routing. However, it may also be the case that they cannot absorb the changes in connection duration

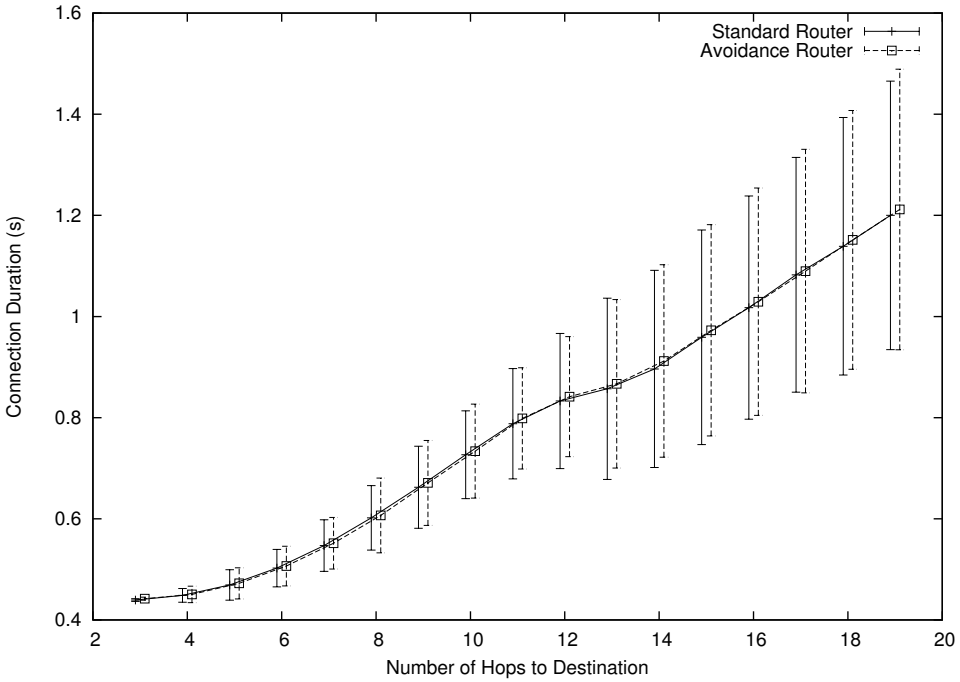

Figure 5.48: Connection duration for avoidance routed packets

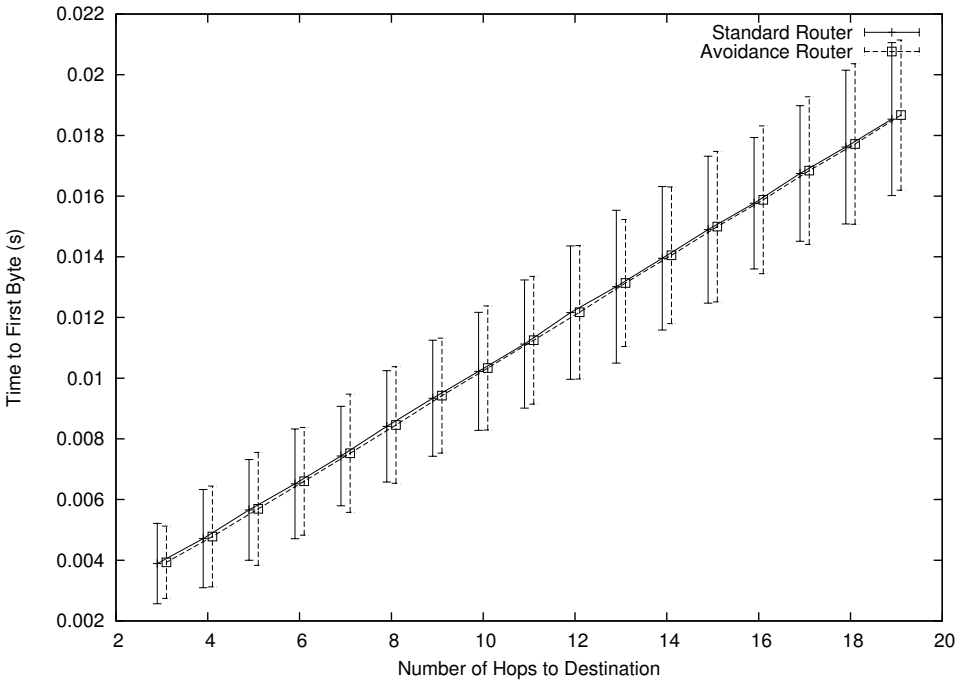

Figure 5.49: Time to first byte for avoidance routed packets

present in the Internet today.

The cause of the increase is due to the extra operations taken by avoidance routing. Avoidance routing takes five extra steps that a standard router does not. As stated previously, both routers look for IP options. After the avoidance router detects the option, it has to determine that it is the avoidance routing option. Then, it must determine what type of avoidance message it is. Once it has determined it to be a forwarding message, it must find the cache ID stored in the packet. The router must look up in the cache based on the ID and modify the ID stored in the packet with that of the next router. Finally, both routers will update the check-sum and forward the packet.

These extra steps can be mitigated if avoidance routing becomes popular. A router could simply assume that a packet may be forwarded via avoidance routing and try to grab the cache ID out of the packet immediately. Since the ID is always at a fixed location in the packet, this would be easy to get quickly. While it is getting the ID, it can simultaneously determine if this is an avoidance packet. This could be sped up and simplified if a mark is placed in the header, such as in the diffserv field. If the packet is not an AR packet, the cache look-up can be aborted or ignored for the standard forwarding look-up (the standard look-up would always be performed in parallel). If the packet is an AR packet, the look-up will already be done and can be quickly forwarded. This would limit avoidance routing to two operations (with many in parallel) instead of the additional five listed above.

We have shown that there is no significant routing delay for packets using avoidance routing in our router. Unfortunately, this is not the same as showing there is no delay for core routers using avoidance routing. This is due to the fact that core routers move at line speed. Without new hardware to support avoidance routing forwarding, all of avoidance routing would have to be handled in software. This would result in every avoidance packet being placed on the "slow path", sometimes referred to as the control or services plane. Core routers can only move packets through the control plane as fast as their processor can handle them. A top of the line Cisco router only contains a quad core 2.13 GHZ Intel

Xeon Processor [Cis12b] which may be able to offer a rate of 1 GB (assuming no other load). By contrast, Cisco has routers that can achieve 140 Gbps, as well as MPLS routers that can achieve the same rate [Cis12a]. This means that avoidance routed packets would suffer nearly a two order of magnitude performance degradation. It should be noted that forwarding using avoidance routing essentially uses MPLS, and avoidance routing's design could be adjusted to make use of these high speed MPLS routers.

While our analysis shows that their is no increase in delay for packets using avoidance routing versus those not using avoidance routing, it does not address the delay introduced via path changes. That is, avoidance routing may use a very different route than standard routing. The path length may be significantly longer, or shorter, than the original path. Unfortunately, scenarios constructed on Deter are not large enough to fully explore this area. Further, the relationship between packet delay and path length is still a open question for research [SSW10], so the effect of increased path length due to avoidance routing is uncertain.

One may be asking what is the path setup penalty? Clearly, a transfer will have to wait for a path to be set up before it can be used, increasing the connection duration. Unfortunately, there is no reasonable way to measure this. The Deter testbed, where our implementation was tested, does not currently support large scale experiments. Without a rich enough set of nodes and avoidance properties, attempting to measure the setup time would not be a fruitful exercise.

We could try to determine this number in small scale and extrapolate to large scale. This is a poor idea. One of the primary overhead penalties for avoidance routing is searching through the stored advertisements. An Internet router has far more advertisements than the scenarios we could construct in Deter. Thus, the penalty we determined would not only grow in terms of path length, but also in terms of number of advertisements.

This penalty will differ based on criteria, source and destination. Our results discussed in Section 5.5 show that the number of nodes visited can be quite high. In these cases,

the setup cost would be high. However, in the majority of cases where the path is found quickly, the setup cost would be low. Further, once the path is set up, any subsequent connections would not pay this penalty at all.

Finally, it is not clear that trying to determine this information is useful. Any avoidance packet will not be transmitted until the path is constructed. Thus, penalties to TCP SYNs and other control packets will be mitigated. Further, the longer the connection duration, the lower the relative setup penalty.

It should not be understated however, that anyone wishing to use avoidance routing must be aware that there will be an initial path construction penalty. If one cares enough, they may be willing to pay a very high penalty. Other security mechanisms, such as Tor, also have initial penalties that users are willing to pay for increased security. Finally, as most paths are found quickly (no search required), we expect that the majority of avoidance routes to incur a minor penalty.

CHAPTER 6

Trusting Security Information

Avoidance routing relies on the information carried in the security-vector to make correct routing decisions. It is important that this information is accurate so that avoidance routing performs correctly. If this information is inaccurate, it is possible that a distrusted node receives data it should not. How to ensure that this data is correct is a critical problem that avoidance routing must solve.

Further, nodes may have an incentive to lie about the security properties. If routers are in control of their own security properties, they could lie in order to get data they should not receive. This is why one's neighbors, not oneself, adds the security properties to the vector. But neighbors may still have an incentive to lie. They could modify the properties such that they are more likely or less likely to get selected during path selection. Further, depending on how properties are determined, they may be able to work together to get another node misclassified, thus leading to incorrect routing.

Nodes could also modify the existing security properties in the security vector. In this way, a node could have an affect on another node much farther along the path. To protect against this attack, we use existing solutions that protect BGP data, such as SBGP [KLS00]. These solutions make tampering with BGP data extremely difficult. Using one of these solutions to protect BGP data should intrinsically protect the security vector. This does not mean a node could not simply remove another's security properties. Neither SBGP or BGPSEC would stop this, only detect this. Since a node without security properties is treated as invalid, this would not lead to an invalid node receiving data it should not.

However, this may lead to a node being avoided that would otherwise be valid

While we have solved the tampering problem, we still have not solved the initial lying problem. In this chapter, we will begin by discussing several approaches that could solve this problem. We will highlight why several of these mechanism would not or could not work. The solution we chose, a consensus approach, will then be discussed in detail. Finally, we will discuss how well our approach works in protecting the security vector.

6.1 Possible Protection Mechanisms

There are several possible mechanisms to protect the security properties listed in the BGP advertisement. These mechanisms include authentication, validation, reputation and consensus. While there may be other mechanisms that we have not considered, these are the mechanisms we analyzed for their use in avoidance routing. Several of these approaches assume the security properties originate from the property holder, rather than their neighbor. In this section we will discuss the advantages and drawbacks of each of these mechanisms. Further, we will highlight why we chose consensus over the other mechanisms.

6.1.1 Authentication

Perhaps the most obvious protection mechanism is authentication. Many schemes rely upon authentication. This system, fundamentally, has each data generator sign their data. Anyone who wishes to use this data authenticates the signature. If the data is valid, it must originate from the generator. If it is not, the data has been falsified or tampered with. In this case, an AS would generate and sign their own security properties since the AS would be unable to lie.

While authentication provides strong guarantees, those guarantees only hold if one can trust the signer. Who would be the signer in an avoidance routing scheme? If the signer is the router (or AS), than clearly we cannot trust the signer. This is the party that may

lie to us, and thus could sign whatever they want. We must trust another party to do the signing then.

One possibility is to have a trusted-platform module (TPM) on each router. TPM is a piece of hardware that allows one to determine the integrity of a platform. The TPM on a router would attest that the router has not been tampered with, or is running a known software version. This software could then attest, with the use of the TPM, other portions of the system. Thus the hardware, not the owner, would attest to the security information.

This seems like a reasonable approach, but one must wonder, what are they attesting? Security properties are not necessarily hardware or software properties. Clearly, the TPM could validate firmware version of hardware type, but what about corporate ownership, or geopolitical location? This information would have to be determined some other way. Known software or techniques could be used to determine this information, which TPM could then validate.

For authentication to work as described, one must trust the TPM manufacturer, the router manufacturer and the security property determination software in order to trust the final signature. Unfortunately, this may be an overwhelming trust burden. Further, an adversary may find ways to trick the TPM, router hardware, router software or security software into signing an incorrect value.

Other problems arise as well. Each router on the Internet would be required to validate a chain of trust for *each* security property in the security vector. These signatures used to sign each property or set of properties may be large, since multiple levels would have to sign any property to ensure its accuracy. The size of the signature (or group of signatures) may affect how long it takes to validate them. The size of the signatures definitely would affect the storage burden of advertisements on each router. Global public key infrastructure would also have to exist so that the data could be authenticated, which in itself may be a non starter [Pri05]. And of course, *every* router would be required to have TPM, which is not currently true on the Internet. This would add a significant deployment hindrance

for avoidance routing. For these reasons, authentication techniques are not an acceptable approach for protecting our security information.

6.1.2 Validation

If we cannot authenticate the security properties, perhaps we can validate them. This approach states that while someone may lie about their security properties, we have mechanisms by which we can validate the properties and detect the lies. The obvious question is, how would you accomplish this? An AS could place their own security properties in the security vector. Each neighbor would then validate the properties they received. If a node lied, it could easily be detected, and possibly blacklisted.

How to validate the properties is an open question. As we discussed in Section 3.4, some avoidance properties may be easy to determine and validate at a neighbor. Others may be extremely difficult or impossible to determine. Further, it may place a great burden on a node if it is required to calculate and validate every security property they know about all of their neighbors. The frequency of these calculations may also require resources ASes are unwilling to spend.

If neighbors do not validate the properties, who does? It may be possible for third party organizations to exist in order to validate security properties. These organizations would collect or determine as much information as they could and then publish the results. Avoidance routing users or ASes could retrieve these results and make decisions based on them. Similar to having neighbors validate the information, it is not clear how easy it would be to collect this information as a third party. Further, it requires that each user or AS trust one or more third party. Assuming that there would be competition, which party or set of parties should you trust? What if a user trusts one set of third parties but routers trust a different set?

The difficulty in creating this solution as well as understanding exactly how it would operate has steered us away from this option. This may still be a viable option in the

future and would not be difficult to add to avoidance routing. The simplest way to do it would be to add the third parties "validation rating" as a security property and use it as an avoidance criteria.

6.1.3 Reputation

A related approach to third party validation is to use a reputation system. This system relies on relative rather than absolute information. Simply speaking, each AS or user could give every other AS a reputation based on past performance. Perhaps an AS has validated some properties or has some other reason to trust the security properties another AS is claiming. Conversely, perhaps one has information that an AS is misbehaving. In both cases, these statements would be used as input to a reputation system that would output a "reputation rating" for any AS in the Internet.

Unfortunately, reputation systems are known to be subject to several types of attacks [HZN09] [KC09] [MG06]. These attacks include simple attacks such as making false reputation claims. More complex attacks including colluding and timing attacks are also possible on reputation systems. Finally, like the third party validation system, there are questions regarding who manages the reputation system. If the managers cannot be trusted, the reputations they determine may also be called into question.

While we do not believe a reputation system is a intelligent approach to solving the trust problem, it may be useful in consort to another system. ASes may be able to submit verifiable information to the reputation system that they are using accurate and valid properties. Users could also send in verifiable statements about information they have learned on different ASes. This reputation could then be used as a security property for avoidance routing. This may help a user discover nodes that they may not want to trust, but were unaware of their behavior prior to using the reputation system. If a strong reputation system can be developed, we definitely believe it should be used to help augment the security added by avoidance routing. However, we cannot overstate that we do not believe

using a reputation system is a viable way of trusting security property information.

6.1.4 Consensus

The final approach we analyzed and chose to use is a consensus approach. Consensus systems leverage the fact that for any one value, there are multiple speakers. Further, they assume that a significant portion of the speakers are stating accurate information. They can use this information to determine the accuracy of the values being stated, or perhaps calculate a "consensus" value. Imagine a room full of temperature sensors, each reading and reporting the temperature in the room. A simple consensus system would average all the readings and report this as the temperature of the room. A more advanced system could use statistical techniques to detect outliers and discard them, preventing malfunctioning sensors from distorting the calculation.

A consensus system for avoidance routing would work in a similar way. Recall that in the design of avoidance routing, each neighbor reports the security properties of a node. Thus, like the temperature sensors, there are multiple nodes all making claims about a neighbor's properties. Using these claims, we can use a consensus system to detect liars.

There are two ways, in fact, that a consensus approach could be applied to avoidance routing. The first is to take all the claims made by neighbors and use them to calculate the "correct" set of properties a router has based on its consensus. Avoidance routing decisions would then be made based on the "correct" information. This approach has a serious drawback — if a majority of a node's neighbors are colluding, they could convince others that this node has different properties than it actually does.

Consider the example in Figure 6.1. In this example, the gray nodes all share a security property. They wish to convince node A that node X does not have the gray property. To accomplish this, all the gray nodes claim that X is not gray. A's consensus system would generally determine this information as being the consensus result, and thus claim it to be correct. X would than use the "correct" consensus result, rather than its claim. In this

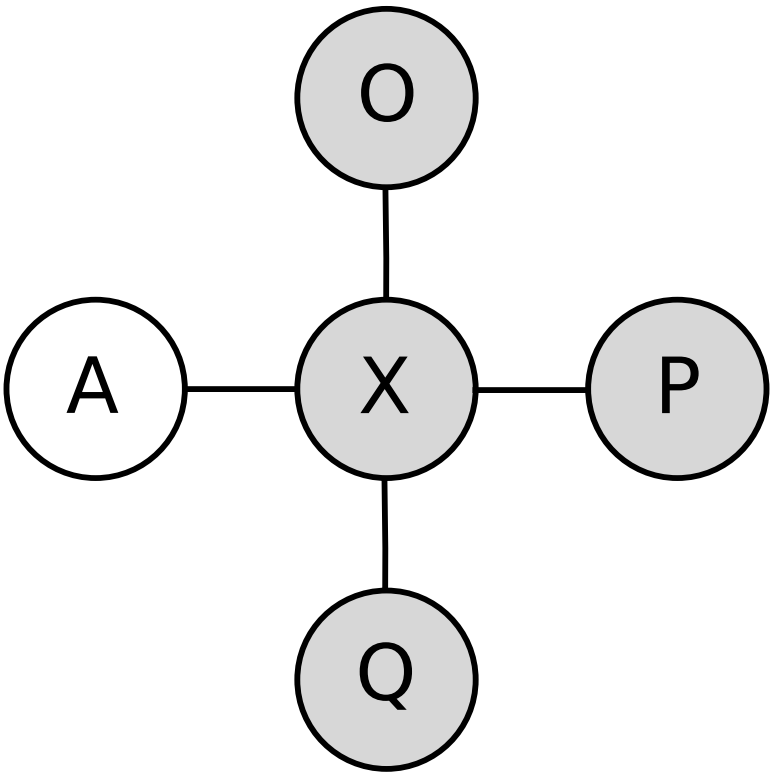

Figure 6.1: A simple topology where colluders may force a incorrect decision

way, if an avoidance request arrives at A to avoid gray, A may give that request to X, when it shouldn't.

This situation may not be unlikely in the current Internet either. Recall that a principle motivator for avoidance routing is distrust between nation states. A large nation may be able to dominate the consensus decisions made at a smaller neighbor. When an avoidance request arrived at that neighbor, it could get incorrectly routed to the larger nation that may have an interest in the request. Do not forget that the United States dominates the Internet graph, dwarfing any of their neighbors.

Avoidance routing must not be subject to this situation. Thus, we do not use a consensus system to determine the "correct" answer. Instead, we use the second mechanism for applying a consensus approach to avoidance routing. This mechanism uses a consensus approach to calculate the *consistency* of security properties, rather than the "correct"

security property. This is accomplished by determining which property has the strongest consensus agreement about it.

Only consistent properties are used. Nodes whose consensus results in a low consensus will not be used. However, it should be made clear that a node will never route to a neighbor *it* believes to be invalid, regardless of what the consensus opinion is about that neighbor. Our consensus system is designed to determine how trustworthy a node is based on their claims. How this is accomplished will be discussed in the next section.

6.2 Using Consensus for Trust

The avoidance routing consensus approach is designed to determine the consistency of the information in the security vector. To accomplish this, we treat each AS as a speaker who states his beliefs about the properties of his neighbors. Since each node knows the claims of many other nodes through the security vector, we can see what other speakers are claiming about their neighbors. Through this mechanism, we can see what the consensus opinion is about an AS's security properties as well as if a node is a consenter or dissenter.

Avoidance routing uses consensus in a couple of ways to protect the security information. The first mechanism used is the one discussed previously, as a way to determine the consistency of the information in the security vector. The consistency of the security information is called **strength of agreement**. Nodes with stronger agreement (more nodes in the consensus opinion) will have more consistent information.

However, avoidance routing also uses the consensus system to determine the quality of a node. This is a reflection of how often a node is in the consensus or in the dissent. A node that is constantly dissenting would have a lower quality, which we call **judgment**. Conversely, a node that is always in the consensus would have strong judgment. A node's judgment can then be a factor in its contributions to the consensus.

How we actually determine the consensus, as well as how it is used in avoidance routing,

will be discussed in the subsequent sections. These will cover what strength of agreement is and how it is calculate. How judgment is determined, as well as how agreement and judgment relate, will be covered. Further, two other factors in the consensus calculation will be discuss, volatility and confidence. Finally, we will discuss how all of this is put together to influence avoidance routing decisions.

6.2.1 Agreement

In order to build a consensus, we require a few definitions. First, we have a **belief** which is a set of properties a node is claiming about its neighbor. An example belief is "node X is in Canada" or "node Y is owned by Pacman Inc." Many nodes may share the same belief about their common neighbor. For a given belief b_i, we define **strength of belief** for b_i to be s_i, which is the percentage of nodes that share the belief about their common neighbor. Formally, we can define s_i as:

$$s_i = \sum^{N_i} \frac{1}{N} \tag{6.1}$$

In Equation 6.1, N_i is the number of neighbors that have belief b_i, and N is the total number of neighbors of the node in question. Figure 6.2 shows a simple example where four neighbors of node X are stating their beliefs about node X. In this example, nodes two, three and four are all stating the belief that X is gray. Node one is stating the belief that X is blue.

According to Equation 6.1, the strength of the belief, X is gray, is 3/4. Similarly, the strength of the belief, X is blue, is 1/4. In this example, the consensus opinion would be X is gray, with the dissent being X is blue. This is how we define agreement, as the strongest belief on a given node. Formally, agreement on node x is:

$$a_x = \max_{b_i \in B_x}(s_i) \tag{6.2}$$

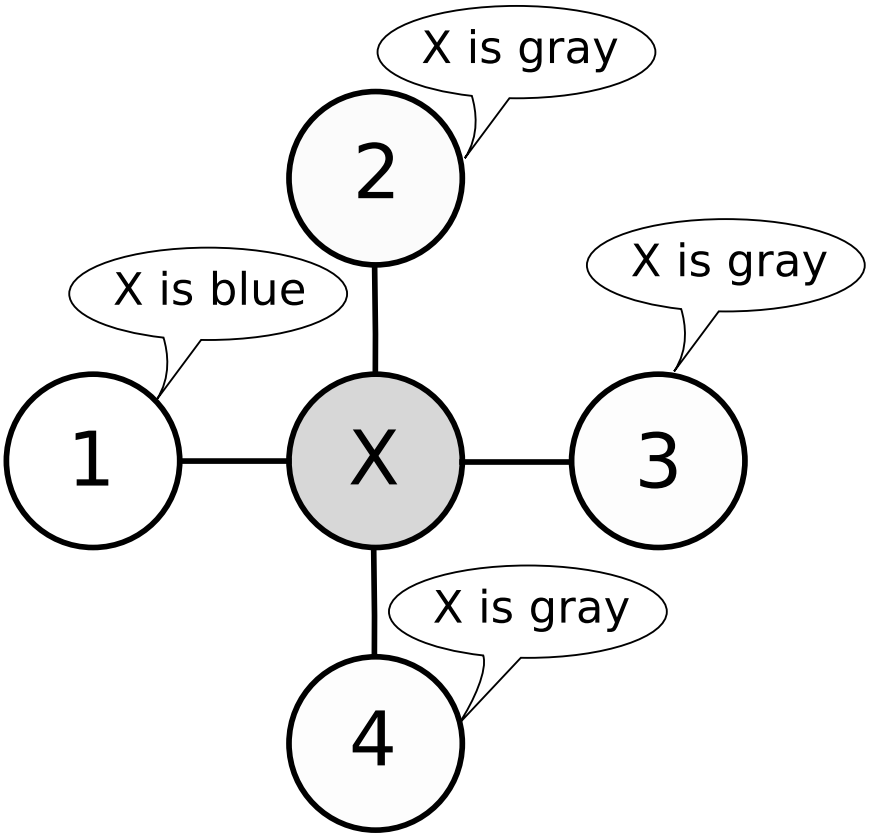

Figure 6.2: Simple topology where nodes are stating their beliefs about node X

where B_x is the set of all beliefs on node x. These two very simple equations allow us to very quickly determine what is the strongest belief about a node. Agreement gives a simple way to quantify the strength of the consensus. Recall that a fundamental assumption for consensus systems is that a significant portion of the speakers are supplying accurate information. We use this assumption as well to determine the consistency of the security properties. Properties with higher strength of agreement are believed to be more accurate, since a larger proportion of nodes supports this claim. A user may wish to only use nodes whose properties are highly consistent. How they accomplish this will be discussed in Section 6.2.6.

6.2.2 Judgment

Agreement is a simple, quick way to try to determine which beliefs are consistent beliefs. However, agreement does not take into account how often a node is correct. It would be useful to know if a node is always a consenter or a dissenter. A node that is constantly in the dissent may represent a node that is acting like it uses avoidance routing but constantly gets properties wrong so that it is never used. Determining how consistent a node is would be very helpful. Given this information, we could weight each node's opinions accordingly. To accomplish this, we introduce *judgment*.

Judgment quantifies how strong a nodes' beliefs are. There are multiple ways to accomplish this. The simplest is to average the strength of belief for all of the beliefs a node has. To formalize this, the judgment of node x is:

$$j_x = \frac{1}{N_x} \sum_{i=1}^{N_x} s_x(i) \tag{6.3}$$

In Equation 6.3, N_x is the number of neighbors of node X and $s_x(i)$ is the strength of node x's belief about its neighbor i. For a concrete example, consider Figure 6.3. In this example, we want to gauge the judgment of node X. Node X has three neighbors — nodes 1, 2 and 3. To calculate X's judgment, we must determine what the strength of X's beliefs are about its neighbors. Each belief about a node is shown on the annotated links. Thus, nodes X, A and B all think node 1 has the security property one. Using Equation 6.1, the strength of belief *one* is 1.0. Nodes X and E are in agreement about node 2, and the strength of belief *two* is 1.0. Finally, X and C are in agreement about belief three on node 3, but D is in dissent. Therefore, the strength of belief three is 2/3. X's judgment is the average of these three beliefs, which is 8/9.

Why is X being penalized even though all its beliefs are "correct"? Further, X is in the majority with all its beliefs. In practice, avoidance routing has no idea whether information is accurate or not. The consensus approach assumes that a consistent belief is an accurate

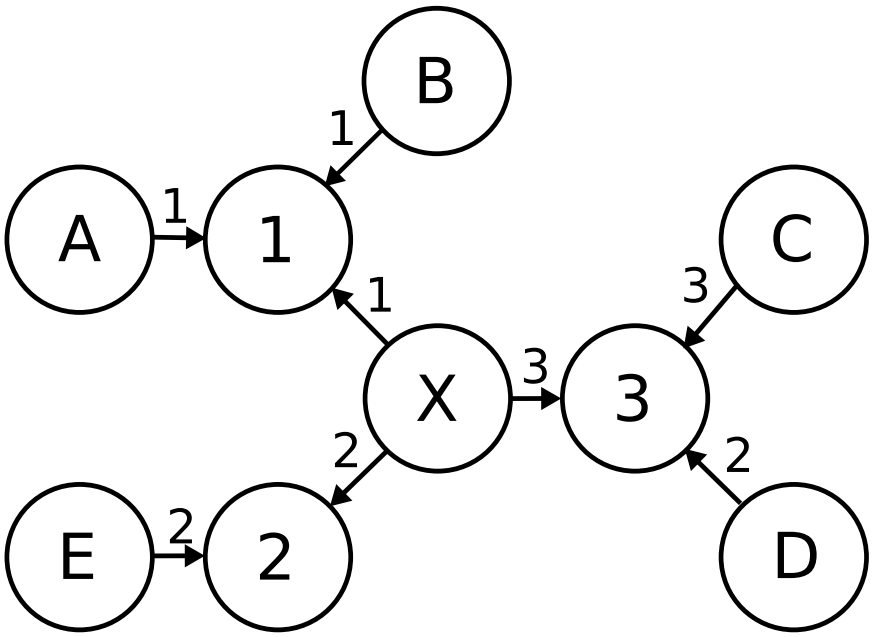

Figure 6.3: In this topology, each letter node is making a claim about a numbered node, where its belief is shown in the annotated link

belief. However, the fact that dissent exists is important information that should not be disregarded. Perhaps node D has new information that X has not yet received. In this case, X's belief is actually incorrect.

Judgment is a powerful tool that may significantly help users determine which nodes they would like to use. A user may only wish to use nodes that have very high judgment (i.e. are always right). Using the same line of thought, a user may wish to avoid nodes that do not have good judgment, as they tend to make "bad decisions". Further, we can use judgment to weight a node's contribution to agreement. Nodes with bad judgment should contribute less to agreement than those with good judgment. We can now redefine strength of belief to include judgment as:

$$s_i' = \sum^{N_i} \frac{j_i}{J} \tag{6.4}$$

In this definition, j_i is the judgment of neighbor i, and J is the total judgment of all

neighbors. We use J instead of N so that even if all neighbors have bad judgment, if they all agree, we still end up with a strength of belief of 1. In our previous example, node X would contribute a judgment of 8/9 to the strength of belief of any of its beliefs.

One with a sharp eye might notice that judgment depends on strength of belief, and strength of belief depends on judgment. If these values are calculated recursively, any judgment less than one will eventually reduce to 0. To solve this problem, judgment is calculated using the strength of belief based on Equation 6.1. In this way, judgment is calculated regardless of the judgment of other nodes.

6.2.3 Normalized Judgment

Judgment determines a nodes quality based on the strength of that node's beliefs. This penalizes a node that lies or is inaccurate about its beliefs. But what if a clever node only lies about one of its beliefs? If a node has a large number of neighbors, such as in the core, this would have a very small effect on that node's judgment. This would minimize the effectiveness of judgment as a useful tool to detect misbehaving nodes.

In order to counter this, we introduce **Normalized Judgment**. Normalized judgment is a very simple concept. When calculating each node's judgment contribution to strength of belief, the node with the highest judgment receives a judgment of 1. The node with lowest judgment receives a judgment of 0. All other nodes are normalized between these two values. If all values are equal, than every node has a normalized judgment of one, and we revert to Equation 6.1.

$$s_i'' = \sum^{N_i} \frac{\frac{j_i - min(J)}{max(J) - min(J)}}{J'} \tag{6.5}$$

Equation 6.5 formalizes the normalization of judgment. Since the normalization can only occur in reference to a specific strength of belief, it is applied to strength of belief. In this case, $\min(J)$ references the minimum judgment value in set J. Similarly, $\max(J)$

references the maximum value. In this equation, J' is the total normalized judgment. This is once again done so that even if all neighbors have bad judgment, if they all agree, we still end up with a strength of belief of 1.

Clearly, normalized judgment penalizes any liar, big or small, significantly more than standard judgment does. How this actually plays out will be discussed in Section 6.2.6. Further, how this evaluates when compared to standard judgment will be shown in the evaluation of the entire consensus system. This will be discussed in Section 6.4.

6.2.4 Volatility

Volatility is a metric that is designed to determine how often a node is moving in and out of the consensus. The idea is that a faulty or malicious node may constantly change their beliefs in order to harm the overall consensus calculation. To calculate volatility, first we need to quantify what it means to "change" beliefs.

$$\delta(b_t) = \begin{cases} 1 & \text{if } s_{b_t} = a_x \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

Equation 6.6 quantifies the change of belief. In this equation, b_t is the belief a node has at time t. This function states that if strength of the belief at time t is the agreement belief, return 1. Otherwise, return 0. A critical question is what does b_t or b_{t-1} represent? As stated, b_t is the belief a node has about a neighbor at time t. The value b_{t-1} is the previous belief this node had about that neighbor. Thus, b_0 is the first belief a node has about its neighbor. Belief b_1 would be the second belief, which only exists if this node has changed its beliefs. If this node never changes its beliefs, it would only have b_0, its original belief.

$$v_x = \frac{1}{N} \sum_{t=0}^{N} \delta(b_t) \tag{6.7}$$

Given this definition for "change of belief", we can define what the volatility of a node is. Equation 6.7 quantifies volatility, which states that for the last N beliefs, what is the average change in belief? If volatility is close to one, this means that each time node X changes its beliefs, it tends to be in the consensus. A value near 0.5 would suggest that this node consistently moves in and out of consensus. Finally, a value near 0 would suggest a node is never in consensus.

While it may appear that judgment and volatility measure the same thing, they actually measure two related, but different, quantities. Judgment is a measure of the percentage of a nodes current beliefs that are in the consensus. Volatility is a measure of how often a node's past N beliefs have been in a consensus. This may be useful when one node detects a change in a property but others have not detected it yet. This node will pay a judgment penalty for being out of the consensus at the current time. However, their volatility may show that generally they are in the consensus and one should not penalize them for having low judgment.

6.2.5 Confidence

One final question still remains, should values of agreement be treated equally? A better way to formulate this question is, given a value of agreement, how confident am I in its consistency? One might be wondering, agreement is a measure of consistency, why are we worried about the consistency of our measure? Assume that we have an agreement of 2/3 for some node. If this node has 3000 neighbors, we may have strong confidence that this measure of agreement is correct. If a node only has 3 neighbors, it is easier to "swing the vote", so to speak. Each node in the latter case has more power over the consensus than that in the former case.

To measure this we introduce **confidence**. Confidence is designed to treat answers from nodes with more neighbors as stronger than those with less neighbors. Mathematically, confidence is simply the inverse of the variance of the beliefs. Confidence is defined as:

$$c_x = 1 - \frac{1}{N-1} \sum^{N} (j_i - a'_x)^2 \tag{6.8}$$

In this equation, N is the total neighbors of x, j_i is the judgment of neighbor i and a'_x is the agreement on node x. As stated, confidence is basically variance, except it is inverted so that values near one are desirable and values near zero are undesirable. It is also important to note that j_i is defined as the judgment of node i if and only if node i is in the consensus. Otherwise, the judgment of node i when calculating confidence is 0. Equation 6.9 formalizes this notion.

$$j_i = \begin{cases} j_i & \text{if } i \in A \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

Confidence can now be used as part of the mechanism to make avoidance routing decisions. Clearly, the equations defined use the standard version of judgment. Versions of these equations exist that can use the normalized version of judgment instead. In these equations, j_i and J are replaced with j'_i and J' respectively, which are their normalized versions. Further a'_x is replaced with a''_x, the version of agreement that is calculated using normalized judgment. It should also be added that we normalize confidence to be between 0 and 1, rather than 0.5 and 1. How agreement, judgment, volatility and confidence are used to influence avoidance routing decisions are discussed in the next section.

6.2.6 Consensus in Use

Avoidance routing uses consensus in several ways. As discussed previously, any conclusion arrived using the consensus mechanism will not override a node's beliefs. This means that other nodes cannot convince a node of a neighbor has a property different than what they believe it to be. All our consensus mechanisms are designed to provide the user of avoidance routing with more options when running avoidance routing. A user may only want to use nodes whose information is highly consistent. Further, a user may only want to use a node

that tends to be in the consensus, thus having good judgment.

When a user makes an avoidance request, they may specify a strength of agreement requirement. This requirement makes avoidance routing only use nodes whose agreement is above the threshold. Thus, another avoidance criteria is the strength of the agreement. Some users may want very strong agreement while others may be more flexible. Nodes not only determine if their beliefs about a node violate the criteria, but also because the strength of those beliefs do not violate a threshold given by the user.

Judgment can be used as an avoidance criteria just like agreement. A user may not want to use a node that is consistently a dissenter. Like agreement, judgment can be used as a security property. A user may only wish to use nodes that have high judgment. Further, another option for a user is to only route through nodes that are in the consensus. Volatility may augment this criteria by allowing a user to specify nodes that tend to be in the consensus.

Finally, confidence allows a user to specify how confident the agreement should be. Since confidence is essentially variance, it can be used by users to not only pick nodes that have strong agreement, but have strong confidence on that agreement. This would reject nodes with only three neighbors who agree at 2/3, but accept nodes with three thousand neighbors who agree at 2000/3000. This, of course, relies on the confidence threshold the user sets. Since edge nodes tends to have only a few neighbors, this may allow one node to perform a DoS attack on other edge nodes. If a user is worried about this occurrence, they may have to specify a low confidence. However, higher confidence may prevent incorrect routing, at the cost of a possible DoS on avoidance routing.

6.2.7 An Example of Using Consensus

To show how our consensus system works in practice, we will run through a small scale example. We will cover how both agreement and judgment will be used in avoidance routing. Due to the small scale of this example, we will not discuss confidence as any

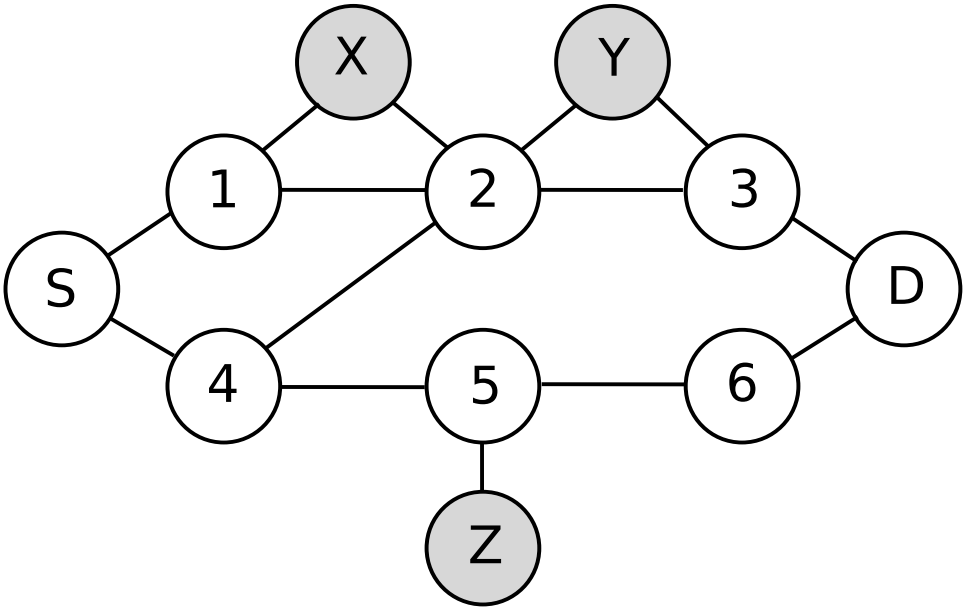

Figure 6.4: A simple topology where S wants to communicate with D

agreement less than 1 will have a low confidence.

Figure 6.4 shows a simple topology where node S wants to communicate with node D avoiding the gray property. Further, it only wants to use nodes with an agreement of at least 2/3. In this example, all gray nodes are lying about their neighbors. We will also assume all nodes have perfect judgment.

When S looks tries to find a route to D, it will immediately see that it has a completely valid route to D via 1. The agreement on node 1 is 2/3, which is acceptable. Thus, it will send the request to 1 and not enter the search phase. Node 1 only has one valid option to D, via node 2. However, both nodes X and Y are lying about 2, and thus the agreement on node 2 is only 3/5. The agreement on all valid nodes is shown in Figure 6.5.

At this point, node 1 has no valid routes to D and must send failure back to S. When S receives this failure notification, it treats it as if the path was invalidated. This is because it perceived the route from S to D via node 1 as completely valid and never entered the search phase. At this point, it will return to the initial phase of the search. It may find that

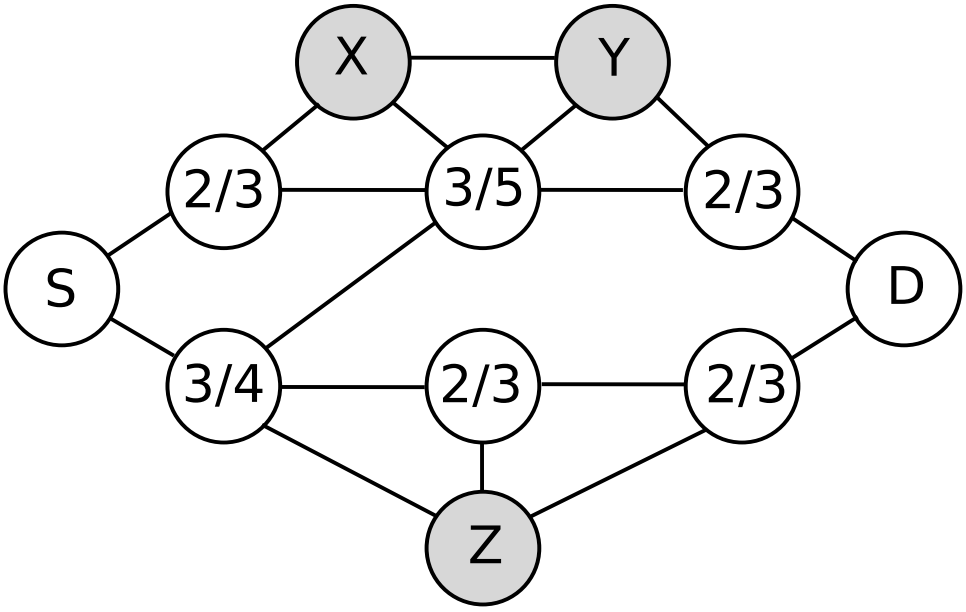

Figure 6.5: The agreement on all valid nodes in the topology

the route through 4 is completely valid and skip the search again. If node 4 had advertised the route through node Z rather than through node 5, it would not have a completely valid path and enter the search phase. Either way, node 4 will receive the request. Note that the agreement on node 4 is 3/4, which is sufficient.

Unlike node 1, node 4 has a valid path to D via node 5. The agreement on 5 is 2/3, which is also sufficient. Thus, the request will go to node 5, followed by node 6 and finally D. When D receives the request, it will generate the success message and send it back along the reverse path.

Before moving on to judgment, let us quickly discuss another example. Assume node 2 is colluding with nodes X and Y. In this example, node 2 says that node X is **not** gray, which node Y corroborates. So what does node 1 do at this point. While the agreement on node X is 2/3 that node X is not gray, since node 1 believes X is gray, it will not use that node. Now, if X and Y do not lie about node 2, node 2 could be given the packet, which satisfies the given constraints. If node 2 is colluding with X and Y, at this point it could simply give the packet to X or Y. This issue is discussed in more detail in Section 7.1 of

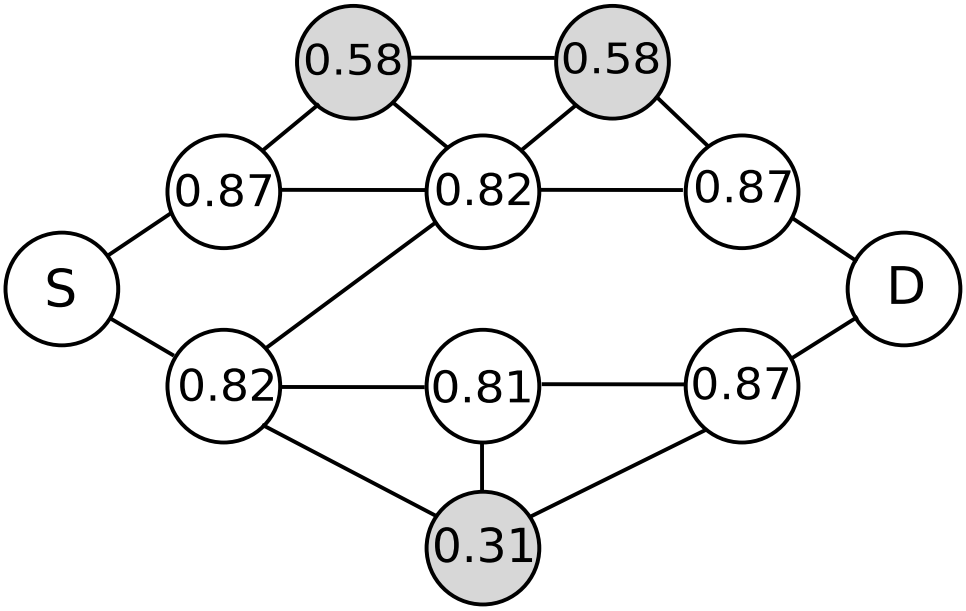

Figure 6.6: The judgment of all nodes in the topology

the next chapter.

Let us change our assumption regarding judgment from the previous examples. Instead of each node having perfect judgment, each node will have judgment relative to their beliefs. Again, let us assume that gray nodes lie about all their non-gray neighbors. The judgment of each node is shown in Figure 6.6. While not shown, S and D have perfect judgment.

If we assume that the sender wants to keep judgment above 0.80, than every non-gray node would still be usable. Thus, judgment of the nodes it self would not effect the routing decision. However, judgment does effect the agreement calculation. The new strength of agreement on non-gray nodes is shown in Figure 6.7

When factoring in judgment, the new agreement for each valid node has increased. This change will have an effect on routing. As one can see, the agreement on node 2 has increased to 0.69, which is greater than the 2/3 threshold. This means that initial choice made by S will be valid, node 1 will use node 2, and neither node will enter the search phase. In this case, node 2 will receive the request and see that node 3 is also valid. The

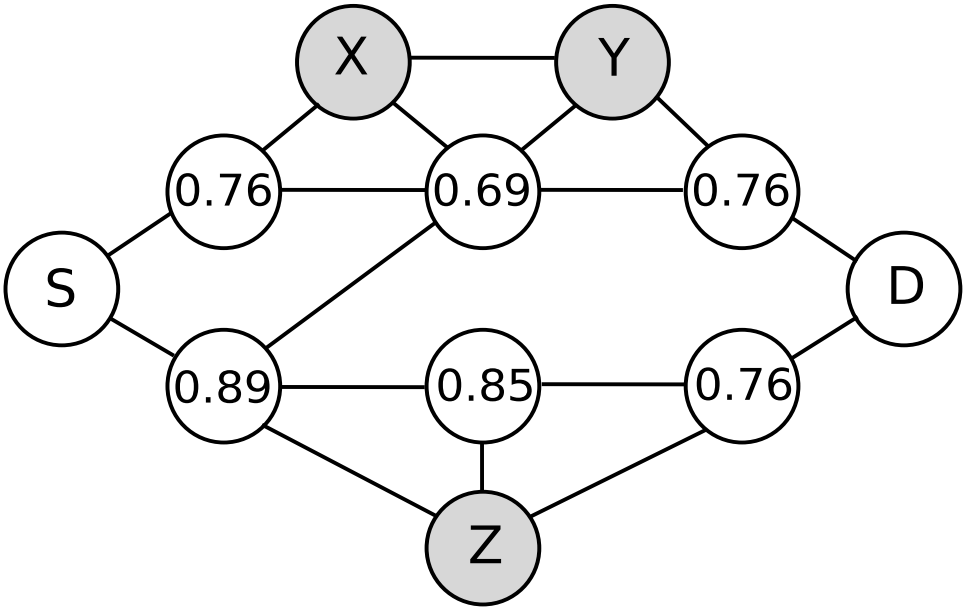

Figure 6.7: The agreement on all valid nodes in the topology with judgment

final route will be {*1 – 2 – 3*} rather than the original {*4 – 5 – 6*} route.

If normalized judgment was introduced, the agreement on valid nodes would increase. This increase is a result of the gray nodes having the lowest judgment. When the judgment is normalized, their contribution to agreement would reduce to 0. Thus, the agreement on all valid nodes would increase. What about confidence or volatility? Unfortunately, the small scale of this example will result in all nodes having low confidence since the number of neighbors is low. We will also not discuss volatility as it requires a history of routing changes, which would result in a very convoluted example.

The final question to ask is, what would be the goal of lying? As we've discussed, it would not allow a malicious party to obtain the data unless a unknown node was colluding. However, lying may prevent a valid node from being used as a routing node. What if the only route to D was through node 2? If X and Y could modify agreement enough such that node 2 is not used, S may be unable to communicate with D. This would result on a denial of service.

6.3 Partial Disclosure

In the previous chapters, we discussed the neighbor-only variant of avoidance routing. While it clearly performs worse than standard avoidance routing, it provides privacy guarantees for different ASes. However, it seems clear that the information required for our consensus system cannot function in the neighbor-only variant. Basically, consensus relies on each neighbor knowing the claims being made by every other neighbor. Further, to calculate judgment, these nodes must know all the claims being made by this neighbor and the agreement on all of this neighbor's neighbors. If the neighbor-only variant is used, none of this information would exist. We still want to provide a mechanism so that a node can learn all the claims about their neighbors, without disclosing this information to all the nodes on the Internet.

To solve this problem, we introduce standard obfuscation techniques. In this case, we replace the properties in the security vector with a hash of the properties in the security vector. The hash will be calculated via a key shared between an AS and all of its neighbors. You generate the hash of a neighbor's properties using their key. Thus, if all neighbors agree on that node's properties, they will all have the same hash.

To determine if their is agreement or not, simply compare the hashes, rather than the properties themselves. This is valid since hashing the same properties with the same key should result in the same value. We do not determine which hashing algorithm to use, as long as all neighbors are using the same algorithm. Any algorithm that is reasonably strong, that is, cannot determine the properties from the hash, should preserve the privacy capabilities of neighbor-only avoidance routing. In this way, nodes can calculate agreement and judgment throughout the entire topology since the hashes will propagate, but the properties themselves will remain secret.

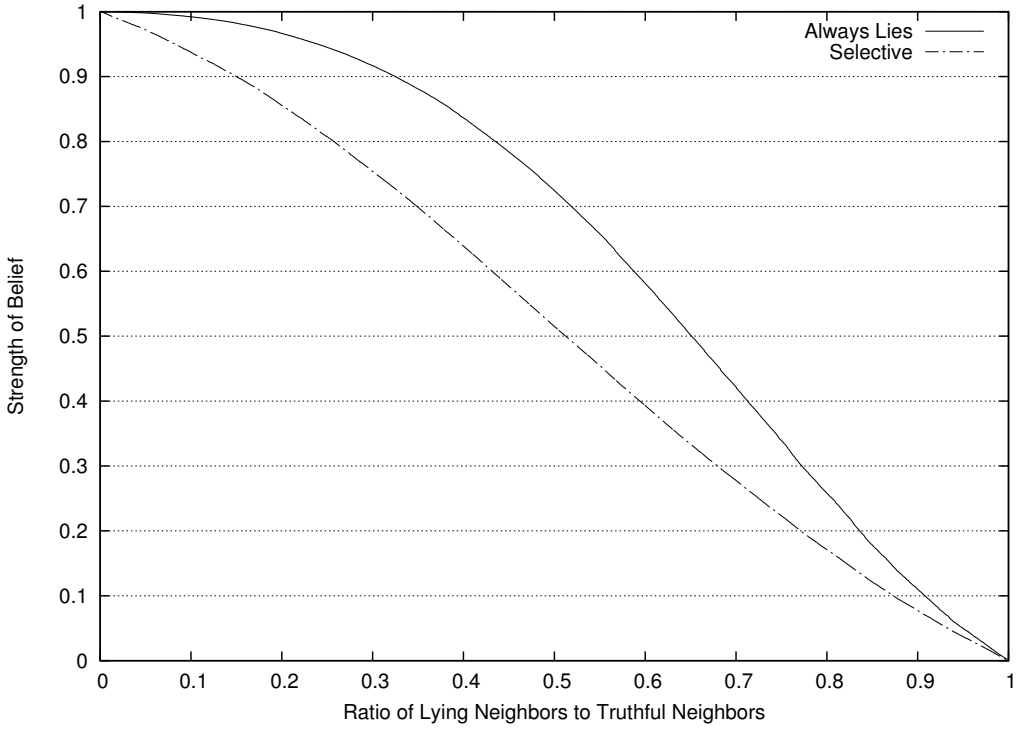

Figure 6.8: Strength of belief as number of dissenters increase

6.4 Evaluation of Consensus

In order to understand how well our consensus algorithm prevents lying, we used our simulator to measure the agreement and judgment of nodes in the core of the AS graph. We increase the number of lying neighbors until all neighbors are lying, and calculate the corresponding agreement and judgment. How the neighbors lie is also varied. In one set of experiments, neighbors lie about every claim they're making. This clearly will have a stronger impact on judgment and thus a lesser impact on agreement. In the other set of experiments, neighbors only lie about the specific node they are targeting.

Figure 6.8 shows how strength of belief for the "correct" belief decreases as the number of dissenters increase. The solid line shows the decrease if dissenters dissent about all their neighbors. The dashed line shows the decrease if dissenters selectively lie only about the target node. Dissenters' judgments are more heavily penalized if they are always in the

dissent; thus the strength of belief is strong even with moderate dissent. Clearly, when nodes are selective about their lies, they take far less of a judgment penalty. This results in a quicker decline in agreement.

One might wonder why at 50% liars, agreement is higher than 50%? This is due to the fact that non-liars are not lying. That is, they are always in agreement and thus will have a higher judgment than the liars. This doesn't reach parity until judgment of liars equals the judgment of the truthful nodes. Since not all nodes have equal number of neighbors, some liars will be more heavily penalized. Truthful nodes will not be penalized until this parity is reached. This is the reason why agreement remains high, especially in the case where liars always lie.

If we apply a little bit of game theory to our attackers, we can try to determine an attacker strategy based on these results. We should assume that attackers only gain a benefit if they can sway a decision, and pay a cost for doing so. Figure 6.8 confirms a standard n of k strategy for attackers. If these attackers want to make others believe a false property about a node, they should wait until they have n colluders so the ratio of n to k passes the threshold of the user. There is no benefit to lying for the attackers if they control less than n nodes. In these cases, they only pay the cost without receiving the benefit of swaying the decision. Since the core is highly connected, it would take a great effort to get enough nodes falsifying their data to get packets mis-routed.

One should recall that the consensus mechanism does not allow a node to modify the claims of another node. The only attack available is to prevent a node from being used. This limits the potential attacks to mainly denial of service. The only other principle attack mechanism is to somehow trick your neighbors that you have different properties. This would be prevented by a user having strong consensus requirements unless the misbehaving node can trick all of its neighbors. Another thing to realize is that the attack strategy listed mitigates the usefulness of volatility. One should be in the consensus until one is ready to make one's attack. However, volatility may still detect faulty nodes.

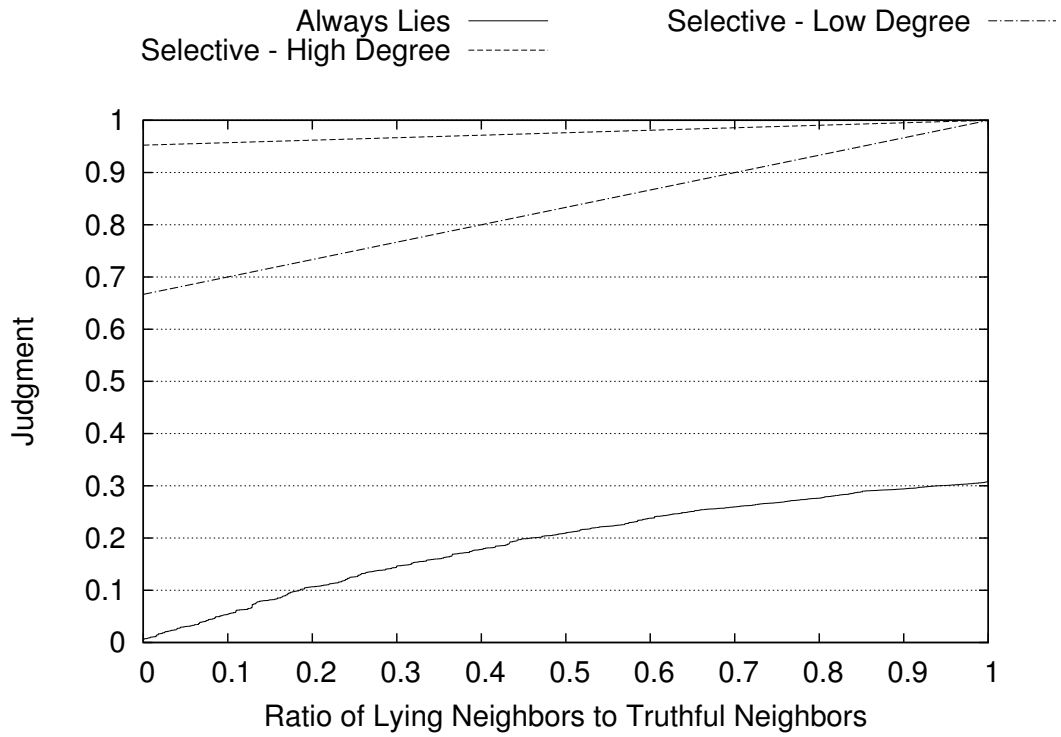

Figure 6.9: Judgment as the number of dissenters increase

Figure 6.9 shows how judgment is affected by dissent. The bottom line shows the change in judgment if a node dissents about everything. The middle dashed line is the effect if a node only lies about the target neighbor and has low degree. In this case, the node only has three neighbors. In comparison, the top line is a node that has high degree. Since judgment is calculated based on a node's neighbors, dissent is clearly affected by this number. This effect is increased when confidence is added, which will be discussed later on.

Normalized judgment may have a large effect on agreement. While it makes no sense to measure normalized judgment itself (since this is based on the agreement being calculated), the effect on agreement can easily be determined. Figure 6.10 shows the affect of normalized judgment on agreement. The non-normalized judgment versions are also shown. In both cases (always lying or selective lying), the effect of normalized judgment results in higher agreement on the correct value. This means that more nodes must lie before the agreement passed below the threshold.

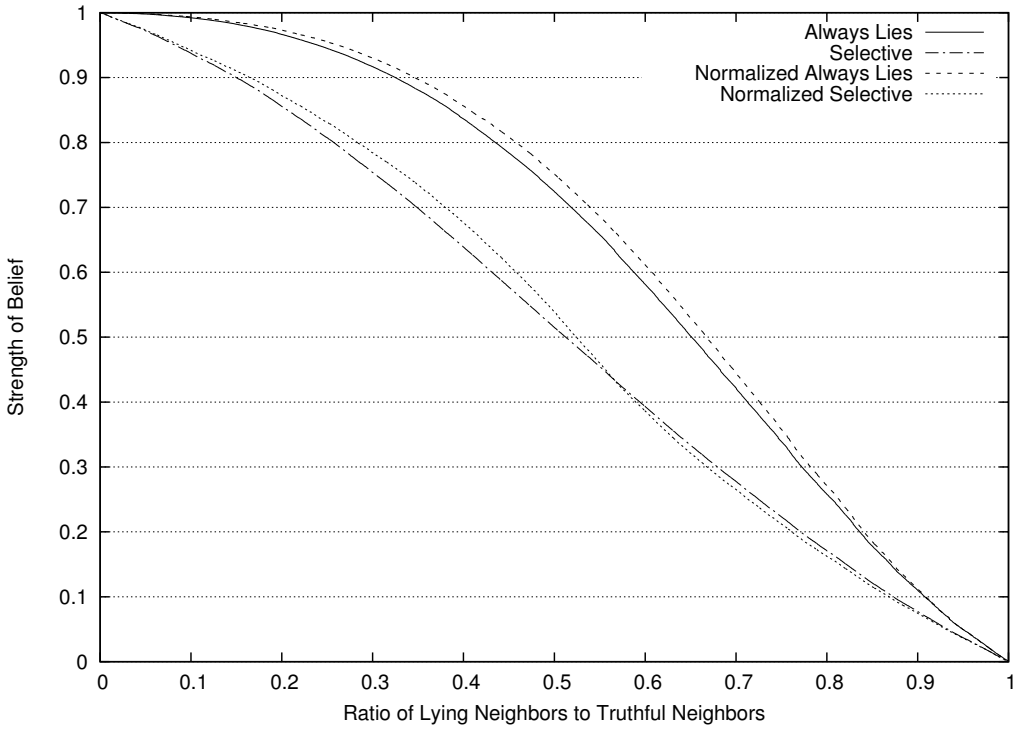

Figure 6.10: Agreement as the number of dissenters increase including normalized judgment

This result is to be expected, as liars are heavily penalized in the normalized version. As one can see, there is an inflection point where normalized selective lying becomes worse than the non-normalized version. This results when a tipping point is reached and the truthful nodes are now being affected by the normalization penalty. We feel this is reasonable as a large number of nodes must lie before this tipping point is reached.

In order to understand how consensus might work in a real-world scenario, we evaluate the effect on agreement if all the nodes in a country lie about nodes outside that country. Figure 6.11 shows the average agreement on "border-nodes" – nodes that border the dissenting country. Each data point represents a different country in dissent, sorted by largest Internet presence to smallest. We also show error bars of three standard deviations beneath each data point. Additionally, we do not show countries that have no access to the Internet through a truthful region. If your only path is through your enemies, avoidance routing will not be useful.

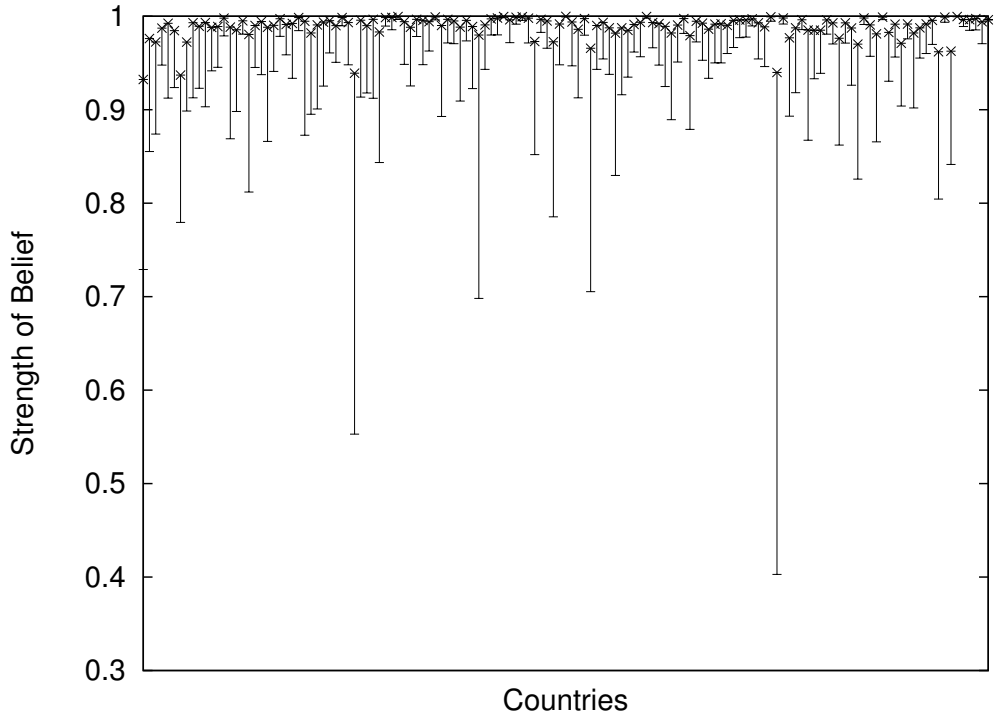

Figure 6.11: Average strength of belief of the neighbors of each country

It is interesting to note that only one country has an error below the 50% point. We note that if a user sets his agreement threshold to 75%, it is statistically negligible that a packet will be misrouted in this experiment. However, it should be noted that this does allow countries to make nodes unusable since they can push the agreement below the threshold. We also see an average judgment for lying nodes of 0.52.

When normalized judgment is introduced, we see an even better picture. The results of using normalized judgment are shown in Figure 6.12. In this example, every node has an agreement of near 100%, and no country has error bars below 65%. This means that, when using normalized judgment, the power to influence avoidance routing for any one nation is incredibly weak. This is even true for super large nations like the US.

The final question left to be asked is what is the effect of confidence? Confidence is a measure of the accuracy of the strength of agreement. Clearly, confidence should decrease as dissent increases. Once the dissenter enter the majority, becoming the consenters, then

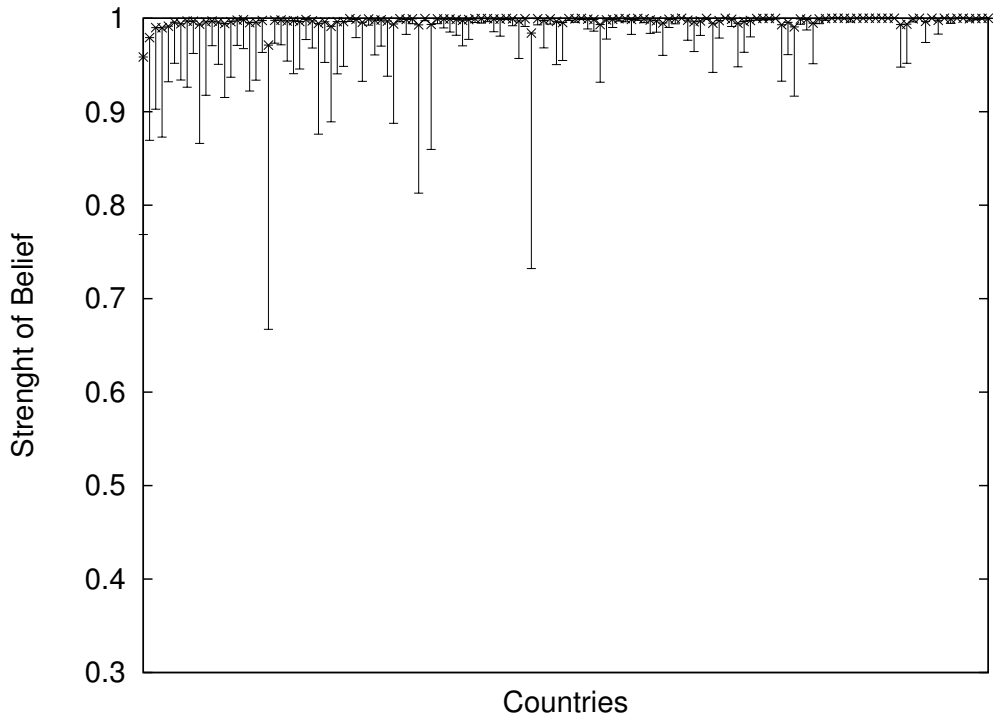

Figure 6.12: Average strength of belief using normalized judgment

confidence will once again increase, believing that this is the correct opinion. Confidence also penalizes nodes who have few neighbors and rewards nodes that have many neighbors. This is all shown in Figure 6.13.

This figure shows the effect on confidence as the number of dissenters increases. Data is shown for a low degree node (three neighbors) and a high degree node (3200 neighbors). Further, normalized and non-normalized judgment is taken into consideration. There are several interesting things to take away from this result. The first is that the effect we expect to see, we do see. That is that a node with few neighbors is penalized more harshly for disagreement than one with more neighbors. Another interesting point is that using normalization decreases confidence. That is, the same effect that causes agreement to be higher also reduces the confidence of that agreement. It is possible to use the confidence calculated by standard agreement and use the value calculated by normalized agreement.

Why does the graph jump? This is caused by judgment. Judgment is a dynamic

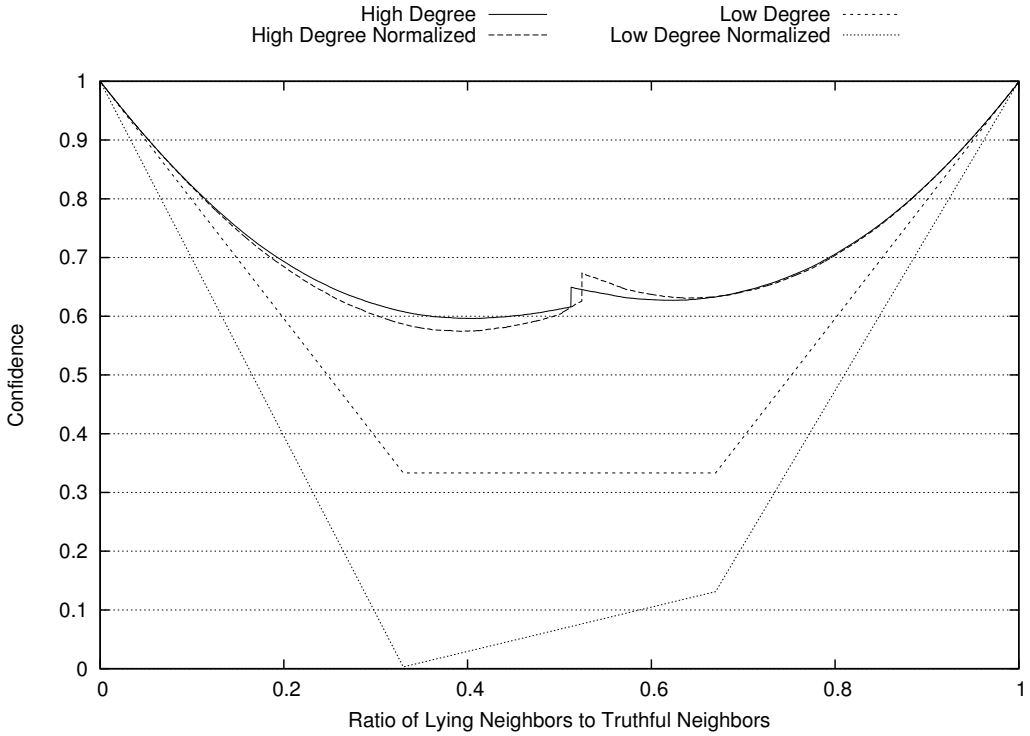

Figure 6.13: Change in confidence as number of dissenters increases

parameter that is changing as the agreement is changing. The jump is caused when the judgment reaches an inflection point where the dissenters have better judgment than the consenters. This inflection point corresponds directly to the same inflection point seen in the agreement data. Further, some neighbors will have more dramatic changes in judgment than others. This is due to the number of neighbors a node has. A node with few neighbors may experience more explosive changes in judgment than one with many neighbors.

Once again, we bring it all together in Figure 6.14. We evaluate the effect on confidence if all the nodes in a country lie about nodes outside that country. That is, how does the confidence on the strength of agreement for border nodes (nodes on the border but outside a country) change when all the nodes in a country lie. Each data point represents a different country in dissent, sorted by largest Internet presence to smallest. We also show error bars of three standard deviations beneath each data point. Additionally, we do not show countries that have no access to the Internet through a truthful region.

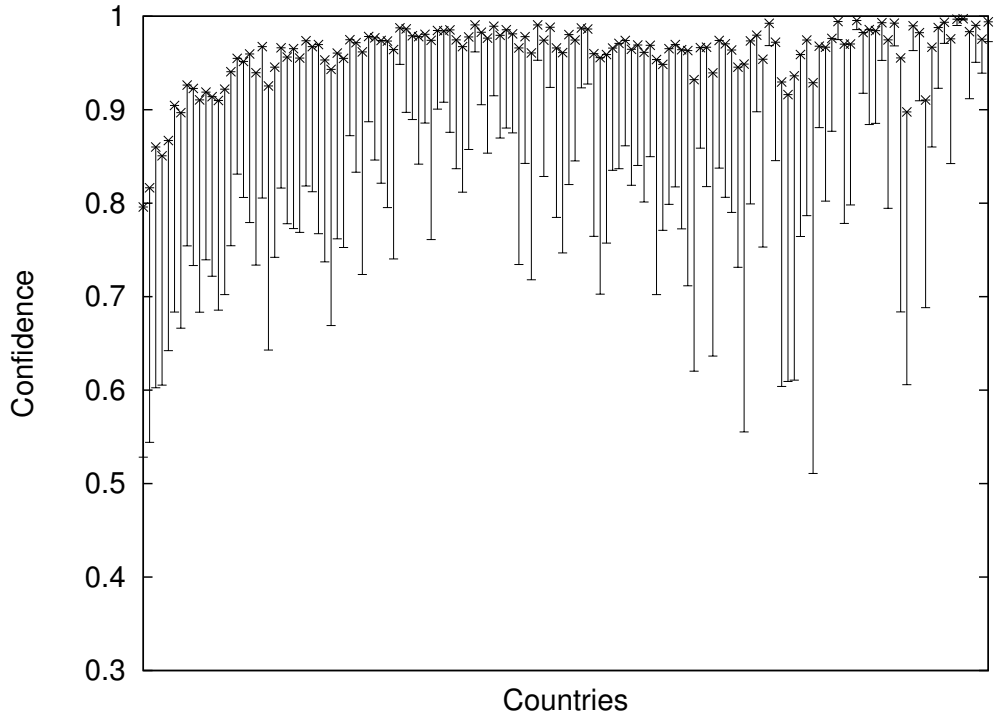

Figure 6.14: Average confidence of the neighbors of each country

The result of this experiment are not quite as strong as those from them agreement experiment. Confidence tends to remain high for nearly all nodes, approximately 90%. Further, 99.7% of nodes have a confidence higher than 50%, as shown by our error bars. This is the worst case for all countries, as many countries do not have error below 60%, and most countries do not have error below 70%. This does mean that, in general, confidence is very strong.

6.5 Limitations

As discussed in this Chapter, we use consensus techniques to attempt to validate the information used for avoidance routing. Agreement attempts to determine the consistency of the information carried in the security vector. Judgment is a measure of the consistency of a node's beliefs and volatility is a measure of the rate of change of those beliefs. Finally,

confidence is an accuracy measure on the consensus system. Our results show that all of these techniques, when used together, produce a robust mechanism for protecting the information in the security vector.

However, our consensus system cannot cover all cases. There are four main limitations to the consensus system. The first involves malicious nodes penalizing legitimate nodes. Since all our consensus values are based on the beliefs of other nodes, these nodes can effect what values another node has. That is, if enough colluding nodes lie, they can reduce a nodes agreement or judgment. This may make this node useless for avoidance routing. It may never be selected since its values are too low.

The second limitation is an extension of the first. Malicious nodes, by selectively adjusting other nodes agreement or judgment, may cause a denial of service attack on nodes further downstream. For example, assume a user wants to send an important message to a destination avoiding Italy. The path to this destination transits a bottleneck node. That is, there is no alternative path to this destination. If this node is next to the Italians, the Italians may be able to lie enough about other nodes to make this bottleneck node's agreement or judgment too low. In this way, the sender would be unable to communicate with the receiver, effectively denying service.

The third limitation involves the situation when a significant portion of nodes are lying. Again, assume a sender wishing to avoid Italy. The request arrives at a router in the UK which begins to search for a valid route to the destination. Perhaps one route to the destination transits France followed by Italy. France and Italy collude together and specify that the route does not transit Italy. The UK router may see this route and believe it has a completely valid route to the destination. Further, it determines that the agreement on its neighbor, France, is strong. That is, everyone believes that the french router is in France.

Luckily, this situation is slightly mitigated by the reverse information. The destination, connected to Italy, would add Italy to the security vector. This information would have to be removed by Italy, which would make the destination invalid since it appears to not be

participating in avoidance routing. In this way, the path would not be selected outright. However, we may decide to still try the french router. At this point, the french router could simply ignore our wishes and route as it sees fit. This is a specific example of a much larger problem, how to trust routers to properly route. This is discussed in the next chapter.

The fourth and final limitation is one which may be the most fundamental. Our consensus metrics assume some stability in the security properties. If the properties are changing quickly, it may result in inconsistent opinions throughout the network. Further, this would completely invalidate our volitility metric. While we currently believe our properties are relatively static, new properties may be introduced which change often. These properties would have to be handled in a different way.

CHAPTER 7

Future Work

How avoidance routing was designed, performs and provides security has been discussed in the previous chapters. However, there are still open questions regarding avoidance routing that are yet to be solved. These include a fundamental question of how does avoidance routing ensure correct routing? Other questions include possible optimization directions such as using partial maps for path selection and conducting multiple searches in parallel.

There are also new research directions available. In our avoidance routing evaluation, we use a geolocation map based on ARIN, etc... data. This data is not perfect, so our map only gives us a plausible geolocation map, rather than a perfect one. One possible research direction is to try and built a better geolocation map. Techniques to solve this problem could also be used as a mechanism for nodes to learn the geolocation of their neighbors.

Another interesting research direction is the possibility of receiver-oriented avoidance routing. Currently, avoidance routing is sender-oriented — the sender sends out the avoidance request and builds the path. While avoidance routing has mechanisms for resolving both sender and receiver constraints, the path construction is still created from the sender. Receiver-oriented avoidance routing would allow the network to know a receiver's avoidance criteria. In this way, a receiver would never receive a packet from a route or region that violates its criteria. Possible approaches to addressing the remaining open questions and new research opportunities will be discussed in this chapter.

7.1 Ensuring Correct Routing

Our discussion of avoidance routing has covered how it works, how well it works and how it prevents lying about security properties. A critical question still remains, how do we make sure that nodes route correctly? While we can detect liars, the system does not prevents an AS from simply giving the packet to adversaries, regardless of the avoidance criteria. Further, ASes could duplicate packets and give a copy to an adversary, while still forwarding along an avoidance route.

In theory, there are three mechanisms to ensure that packets are routed correctly. In practice, one mechanism is basically impossible, one mechanism may be impossible and one mechanism is plausible but difficult. The first mechanism is to construct the Internet infrastructure in such a way that it is impossible for a packet to get mis-routed. The hardware and structure of the Internet guarantee that packets are correctly routed. Unfortunately, no such hardware exists and it may be impossible for any hardware to provide such guarantees. Further, it would require a complete redesign of the Internet. While this may be possible in the future, for the goals of the Internet, this idea is considered impossible.

The second mechanism is to confirm that packets are routed correctly through observation. This works by observing packets as they transit routers and detecting where they leave. Any misbehaving router would then be avoided in future avoidance requests. The final mechanism is to ensure correct routing through construction. This operates by having a user construct their avoidance criteria such that no misbehaving router could ever be used. How both of these work in more detail will be discussed next.

7.1.1 Observation

Observation functions by watching routers and verifying that packets are forwarded correctly. In reality, one cannot "watch" routers. Instead, one would use the previous and next router on the path to determine how the router forwarded. The previous router would

inform us that it gave the node in question the packet, and the next router would determine if it received the packet it was suppose to. If not, it would inform us of the routing error.

Unfortunately, this design is fraught with peril. How often do you test a router? If you test infrequently, you might not catch a router misbehaving. If you test too often, you may over burden well-behaved routers. Further, routers may be selective on what they mis-route, only mis-routing high value data. This would make it more difficult to detect a misbehaving router.

This strategy also requires that all routers are participating. This assumption may be reasonable, as non-participating routers could simply be avoided via avoidance routing. Other problems still exist however. How the validation information is procured is an open question. If the information transits the Internet digitally, it is subject to being dropped or tampered with by the same misbehaving nodes. If it does not transit the Internet, the mechanism for gathering this information would, in itself, be an open problem. Also, a node could simply lie about the information, incriminating an innocent neighbor. How to protect and validate the information itself would another problem to be solved.

Perhaps the final nail in the coffin of observation is that node can misbehave and still correctly route. A node could send the packet to the adversary, who processes it or copies it before sending it back to the original node. The original node simply forwards it correctly from there. From our observation strategy, this extra hop would not be observed. Further, the node could simply duplicate the packet and send a copy to the adversary, avoiding any possible detection mechanism using packet timing. While we could ask the adversarial node if they received the packet, they would simply lie since they have no reason to implicate their colluding neighbor.

7.1.2 Construction

Observation is clearly not a viable strategy for guaranteed correct routing and a complete Internet redesign is definitely out. Ensuring correct routing through proper path construc-

tion is the only viable means forward. This method provides correct routing by placing any misbehaving node in the avoidance criteria. This works under the assumption that a misbehaving node is, by definition, an untrusted node and thus should be avoided. However, this requires that a user knows every property they distrust. This solution has the benefit of ensuring correct routing by construction, but only if that the construction is correct.

Unfortunately, this is the only solution that can possibly enforce correct routing. While we do not feel that this is the best answer, or even a good answer. It basically assumes the end user has already solved the problem. However, it is currently the only answer that we have available. Future work in this area would include investigating better observation techniques than we have considered, or perhaps a whole new mechanism. If better mechanisms are designed to determine misbehaving routers, their identities could then be placed in a database. When a user wished to use avoidance routing, they could consult this database in order to add ASes to their avoidance criteria which misbehave and have not already been added by their other criteria.

7.2 Search Optimizations

Our search performance results, described in Chapter 5, show that avoidance routing performs rather well. However, this is only the general case. There are times when avoidance routing performs exceptionally poorly. Examples include when avoidance routing finds a path 250 nodes long, or visits an extra 2,000 nodes. While these results occur infrequently, even when avoidance routing is performing well, it may still visit an additional 50 nodes on average. We would like to investigate other mechanisms for optimizing the search. In this section we will discuss two possibilities — partial map construction and parallel searches.

In order to understand why an optimization may be effective, we must first discuss why avoidance routing may perform poorly. For this purpose, consider the topology shown in Figure 7.1. The cloud in the topology is a large sub-network of the graph. This topology is

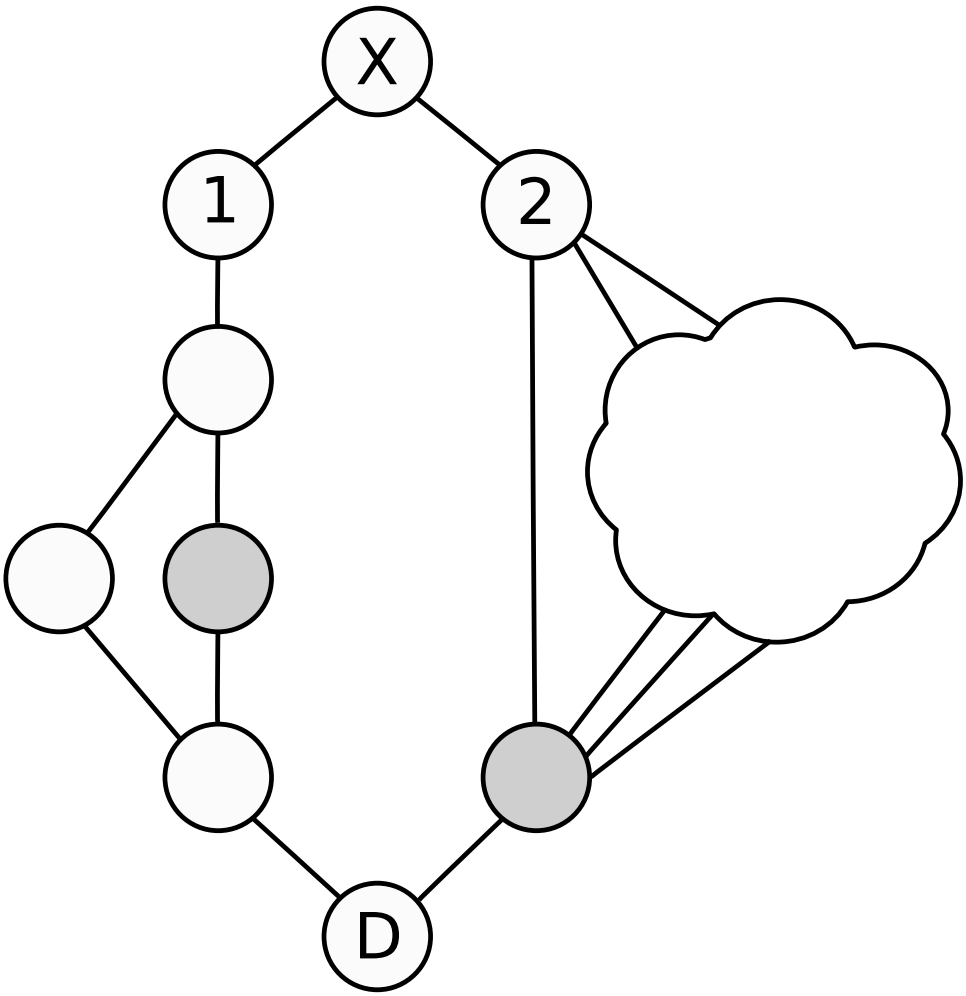

Figure 7.1: Simple topology illustrating when avoidance routing may perform poorly

hierarchical, thus, a node "above" another is the provider of the other node. Our analysis here will assume a "no-valley" policy is in use.

In this example, node X is attempting to find an avoidance route to destination D avoiding gray nodes. Node X has two choices, through neighbor 1 or neighbor 2. Neighbor 2 has a shorter route to the destination and would get chosen by that heuristic. Unfortunately, none of the routes neighbor 2 has will reach the destination. Further, since DFS penalizes bad decisions, all of the paths from neighbor 2 to destination D will be explored. In general, this means that the large sub-graph shown will be fully explored before neighbor 2 reports

failure. This is the principle way by which extra nodes may be visited.

Long paths may occur using a similar mechanism. The primary difference is that some long, obscure path, through the sub-graph shown, will reach the destination. This path will be discovered and used, rather than the short path through neighbor 1. Before discussing how our optimizations might work, it should be noted that exceptionally long paths may be the result of errors in the policy inference of our AS map provider, and not a result of avoidance routing. Even if this is the case, the possibility to visit a large number of extra nodes still exists, and should be minimized.

7.2.1 Partial Map Construction

Partial map construction leverages the fact that there is a significant amount of information available to any one node that is not being utilized in the path selection algorithm. Specifically, each node is aware of a significant portion of the Internet AS topology through the set of BGP advertisements they hold. However, only a specific advertisement to the destination is considered at a time. The remaining information is essentially discarded.

An example of how this information may be used is shown in Figure 7.2. In this example, node X is once again trying to find a valid route to destination D. However, rather than just considering advertisements to D, X has built a partial map out of all of its advertisements. In this example, advertisements from neighbor 1 are shown as solid lines, and advertisements from neighbor 2 are shown as dashed lines. When viewing the partial map, X notices that neighbor 2 advertised a route to A through node N. Further, neighbor 1 advertised a route to D through N. This means that, while unadvertised, neighbor 2 can reach D through N. Thus, X may have a much higher chance of reaching D by using neighbor N.

While this is a simple example, this example highlights how a partial map may be useful. It is important to note that routing through node N may be invalid (as described in previous chapters), but we believe this to be unlikely in the general case. Nodes have multiple valid

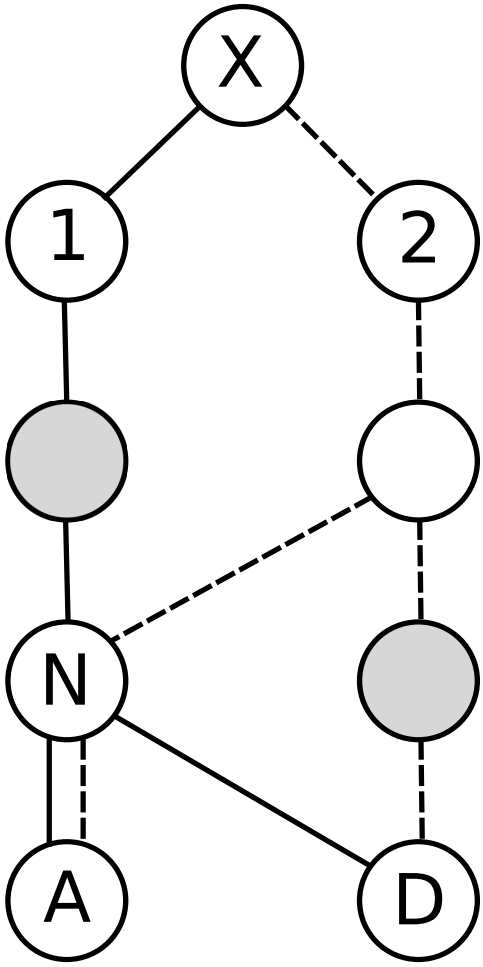

Figure 7.2: Topology illustrating information gleamed from multiple advertisements

routes and we believe that choices for which route to advertise may not preclude other routes. There are also other ways partial maps may be useful. In the example shown in the previous section, Figure 7.1, partial maps may have detected that there were many routes that all had the same failure. This detection may have lead the selection algorithm to pick different routes, assuming the common failure to be a bottleneck point.

Partial maps are an intriguing area for future research. Clearly, they may lead to significantly better path selection. How, exactly, partial maps would be incorporated and how well they perform is what is left to be determined. One could construct the map and run Dijkstra's algorithm to find plausible existing routes. Then, one would simply choose

between them. There may be other ways to use partial maps as well. We feel partial maps to be the best course for optimizing the avoidance routing search, and may have applications in other routing areas.

7.2.2 Parallel Searches

The other obvious approach to improving avoidance routing performance is through parallel searches. The basic reasoning is that running multiple searches in parallel will find better paths more quickly. Basically, this would result in avoidance routing becoming a breadth-first search. Unfortunately, parallel searches have several drawbacks.

The first problem with parallel searches is that they will visit more nodes by definition. This may be minimized by synchronizing the searches. That is, if one search finds a path to the destination, the other searches are terminated. One could accomplish this by adding extra messages to avoidance routing that informs the other searches of success. However, these stop messages may be constantly "chasing" the other search messages. Whether they are able to "catch up" would be a question to be explored in future work. Further, one could have synchronization points where all searches stop to see if anyone has found the destination. Unfortunately, this may excessively delay path creation.

Parallel searches may also expend more resources in the cases when paths are found quickly. As discussed in our results, the majority of paths are found within the first two hops. Parallel searches would be spending extraneous resources for no gain in these cases. Again, one could synchronize the search or use stop messages, but the problems described above would have to be overcome.

The final problem with parallel searches is determining who conducts multiple searches. Clearly, every node cannot conduct multiple searches, as this would result in flooding the Internet. A node with many neighbors may be the best option, as they would be more likely to have a valid alternative. Perhaps a node closer to the source? Does only one node conduct parallel searches? All of these questions must be answered in future research.

Finally, this may open up the possibility for an amplifying DoS attack. Future work in this area would have to mitigate this possibility.

7.2.3 Other Search Algorithms

Avoidance routing uses heuristic-based depth-first search because it is a simple and fast algorithm that tends to find paths quickly. Unfortunately, as we've shown, it can sometimes result in very long paths. There are other search algorithms, besides parallel searches, which may provide better results. The foremost amongst them is breadth-first search (BFS). BFS would have the property of always finding the shortest route. Unfortunately, BFS is difficult to run in a distributed environment. BFS operates by fully exploring one "level" (all the neighbors of the previous level) before moving on. This results in having to synchronize multiple simultaneous searches, which may require a high amount of control overhead.

Another option is depth-first search with iterative deepening (DFSID) [Kor85]. DFSID operates similarly to both DFS and BFS. In DFSID, a maximum depth is set. The graph is searched in a depth-first manner but stops if the depth is reached. If the destination is not found, it increases the depth and tries again. The increasing depth mechanism makes DFSID search the graph in a breadth-first manner. In fact, it is shown that DFSID has the same time complexity as DFS but a far better space complexity than BFS. Using DFSID for avoidance routing may help improve the path selection, but may take longer to actually find paths as the iterative deepening process is undertaken. With some additional stored information, DFSID may be ran without having to visit nodes multiple times.

There are many other graph search algorithms, such as A* or IDA* [Kor97]. The principle problem with A* is it is a heuristic version of Dijkstra's algorithm. This means that A* has the same coordination problem that BFS has for distributed systems. IDA* solves some the memory usage problems of A* but would still suffer the distribution synchronization problem. In fact, many optimal search algorithms for graphs suffer from this same problem. Regardless, an obvious direction for future work is try out several of these other

search algorithms to determine the benefits and weaknesses of each one.

7.3 Geo-mapping

The geo-mapping problem is that of constructing a better geopolitical map of the Internet. The geopolitical map we use for avoidance routing is based on meta-data for each AS obtained from sources such as ARIN. This data is based on registration information or corporate headquarters, and not necessarily the actual location of the AS. While it gives a reasonable approximation for the purposes of evaluating avoidance routing, it would be useful to have a better map of the Internet.

This is currently an open research problem that several people are investigating. Approaches include vector-space distances [HR], triangulation [KJK06] [MO09] and network distance [NZ02] approaches. All of these approaches show some promise, and many show significant limitations. One of the fundamental problems is that routers behave differently than end hosts. Getting routers to respond quickly and accurately to probing messages is a difficult task. Trawling through registration databases may be a way to augment the information determined. This area is still wide open for research. An answer to this question would be useful for further evaluation of avoidance routing, as well as similar technologies.

7.4 Receiver-Oriented Avoidance Routing

Receiver-oriented avoidance routing is the last, and perhaps most interesting, future research direction. Currently, avoidance routing is sender-oriented. When a sender wants to use avoidance routing, they send an avoidance request to the destination, who responds with the success message. If the destination has their own criteria, the success message unions the criteria and builds the reverse path. Finally, the sender must build a new forward path with the unioned criteria.

What if the receiver does not want to receive *any* data from a specific AS or region? Perhaps this region has been conducting a DoS or infiltration attack on the receiver. This capability does not currently exist in avoidance routing. Further, the capability is only generally available at last hop providers, which may provide little benefit.

Receiver-oriented avoidance routing would provide the capability for receivers to specify properties of routers they do not trust, and be guaranteed that they would not receive data through these routers. How might this work? If we assume the deployment of sender-oriented avoidance routing, this becomes a little more tractable. In this situation, the receiver would propagate their BGP advertisements using avoidance routing. They would specify their criteria as avoidance criteria and use avoidance routing to limit the exposure of their advertisements. In this way, untrusted regions of the Internet would never receive an advertisement to the destination, and thus would not even know *how* to route to the destination. These routers may infer that a path exists to this destination even though they do not know about it. It would be up to edge nodes in the trust network to know if a neighbor was attempting to cheat or not and enforce the routing constraints.

This approach is similar to BGP communities, which restricts how advertisements traverse the network. BGP communities provides AS restrictions, while receiver-oriented avoidance routing would provide broader constraints. How well receiver-oriented avoidance routing would perform is an open question for future work. What if sender-oriented avoidance routing did not exist? What mechanisms could be used to perform receiver-oriented avoidance routing? How well would these mechanisms perform? These questions are intriguing and should be answered in the next round of avoidance routing research.

CHAPTER 8

Related Work

Avoidance routing leverages, utilizes and enhances several areas of previous work. These include uses of search for routing, trust-based routing techniques, policy-based routing, label-based forwarding and consensus computation. How we utilize and differ from these techniques will be discussed in detail in this Chapter. Further, there exist alternate approaches attempting to solve the avoidance routing problem. Many of these approaches suffer significant limitations which avoidance routing attempts to address. A discussion of each alternatives advantages and disadvantages will be discussed as well.

8.1 Alternate Approaches

There exist several other approaches that attempt to avoid untrusted nodes. These approaches range from source routing and vector-based routing approaches. Some research has also gone into solving the challenges to deployment. While we have discussed source routing in detail in Chapter 2, we will discuss the other approaches in detail here.

The critical problem for avoidance routing is discovering a valid route to a destination. Zlatokrilov and Levy [ZL07][ZL08] attempt to solve this problem by creating a virtual coordinate system using distance vectors. They retrieve distance vectors from specific reference nodes. Using these distance vectors, as well as the known position of the nodes relative to the source and destination, they generate a coordinate system. Use this coordinate system, they find nodes maximally distant from the nodes they want to avoid. To actually forward the packets, they utilize source routing.

While their virtual coordinate system shows significant promise, there approach has some limitations. To perform avoidance routing using their approach, senders would already need to know the security properties of every router. The authors also do not discuss how to use their system in the current Internet. This may require solving the topology problem discussed in detail in Chapters 2 and 3. Further, a forwarding approach that does not use source routing would be required.

Khuruna et al. [KGA07] use a similar technique to Zlatokrilov except in ad hoc networking. Again, their primary purpose is to find paths that are farthest from the adversarial nodes. This goal is exacerbated by the eavesdropping potential of wireless communication. Unfortunately, their technique suffers similar limitations to Zlatokrilov's, while also incurring a slow convergence time.

A similar approach for solving a slightly different problem is the (Wormhole-Avoidance Routing Protocol), proposed by Su et al. [Su10]. WARP is designed to combat wormhole attacks, a mechanism for making it appear that one has a very good route to a destination. This mechanism will cause nodes to get selected more often in the routing algorithm, effectively observing all the traffic. WARP combats this by sending out multiple AODV route requests and receiving multiple answers. It then determines the distribution of path attributes, runs a frequency analysis, and excludes routes that deviate from the norm. Finally, it picks one of the satisfactory routes to use.

Their approach to avoiding malicious nodes is designed to solve a slightly different problem than avoidance routing, making its applicability limited. Specifically, malicious nodes will receive the route requests, and thus be fully aware of the potential communication. Avoidance routing prevents this by never giving the avoidance request to invalid nodes. Further, changing network conditions can significantly hamper their frequency analysis.

Another alternative is to use an overlay network. Overlay networks have been used for a variety of purposes, including multi-cast routing [BBK02][CRZ00], content replication [CHM02][Aka08], content distribution [CRB03][LPR04][Gnu08], and privacy and

anonymity [Tor12]. In general, two types of overlays could be used to accomplish our goal. A generic overlay network could be used, where the overlay nodes are set up to avoid the routers we distrust. This requires the creator of the network to be positive that the paths between nodes do not go through untrusted routers. This may be extremely difficult if the avoidance criteria is sufficiently complex. The overall topology problem may also have to be solved in order to guarantee proper avoidance routing.

Overlay networks also suffer from other limitations. Unless routing is static, routes between overlay nodes may change, causing data to go through untrustworthy ASes. Furthermore, overlays are more fragile than our scheme. An attacker may only need to initiate a DoS attack on a small number of overlay nodes in order to bring down the network [AS07]. Finally, setting up an overlay can be costly, which may provide a strict barrier to entry for an avoidance routing solution.

An anonymity-type overlay network like Tor [Tor12] could be used to provide avoidance routing. These networks use encryption and onion routing [And96] to hide the identity of the sender from the receiver or anyone intercepting the communication. In principle, being able to determine who is talking to whom requires compromising the entire network. The problem with these networks is that they can be heavyweight and easy to detect [MD05]. An adversary country could kill Tor traffic, denying service at a critical moment. This approach also assumes that the attacker cannot break the cryptography, which is an assumption we prefer not to make. Even without breaking the cryptography, there are several mechanisms by which one can use to de-anonymize the communication. Several attacks exist that rely on timing and correlation to determine who is communicating [ZFG04] or what type of communication they are having [WCJ05]. How to protect tor from these attacks is an area of ongoing research [JFS10] [ZYF10]. These limitations suggest that Tor, or networks like Tor, may not be a good solution for avoidance routing.

A final alternative approach would be to design a completely new routing protocol. In this case, a link state routing protocol would be optimal, and would be able to support

many different types of security parameters, such as country, router type, etc. Each node in the link state graph would specify all of its parameters. When avoidance routing was conducted, any node that did not meet the requirements would be removed from the graph. Then one would simply run Dijkstra's algorithm [Dij59] on the graph to find a path to the destination. While this solution is viable, it relies on being able to get global information about the ASes on the Internet, which we believe is infeasible. Further, it requires the deployment of a new routing protocol, toward which the industry has shown reluctance.

Zhang et al. follow the idea of a complete new routing design in their SCION proposal [ZHH11]. Scion is a future internet architecture that can support avoidance routing concepts. In SCION, the Internet is subdivided into trust domains (TD) that share common properties, like geopolitical location. Each destination specifies a set of TDs from the Internet core that they are willing to use. When a route is constructed from source to destination, the source picks the TDs it will use to reach the core, and selects from the TDs out of the core that the destination has specified. This design is quite promising but does not have the same flexibility for solving the avoidance routing problem as our solution does. Their solution does show significant promise as a general trust-based routing solution. The major limitation of any future architecture is it cannot readily be deployed.

While not a direct alternative to avoidance routing, Laskowski et al. discuss the challenges to deploying any user-directed routing scheme [LJC08]. They define user-directed routing as a mechanism where a user has some or full control over routing, which avoidance routing provides. They conclude that a market-based approach may allow deployment of user-directed routing as long as the payment system is flexible, which we agree with. Further, they argue that control is a key reason ISPs shy away from user-directed routing. They argue that if price is tied to route, then ISPs may be able to maintain their control. This may be less of an issue for avoidance routing. While users inform ASes of their criteria, ASes get to make the final routing choices based on their policies (as long as that does not violate the criteria). Avoidance routing may also be able to benefit from the flexible

pricing structure they discuss.

8.2 Trust-Based Routing Protocols

There are several proposed trust based routing protocols. Most of these protocols work in a rather simple manner. Trust values are assigned to each node and nodes whose values are invalid are not used for routing. There are some other protocols which take a more active approach by determining trust on-demand.

In the ad hoc routing, Yi et al. [YNK01] propose a routing protocol to avoid untrusted nodes. This protocol assigns each node a trust value a priori. Each trust value has a specific encryption key associated with it. Messages are then encrypted based on trust level, where nodes with higher trust levels can decrypt and forward, while nodes with lower trust levels cannot. This technique is not practical for the Internet. Each node would have to have different trust information for different groups of senders, and the cryptography is too expensive for routers to process at line speed.

Another approach, the Trust-Aware Routing Protocol (TARP) [AKB06] is close to ours, as decisions are made based on route attributes rather than global trust level. TARP avoids nodes based on power consumption or software capabilities, not necessarily security properties. TARP is based on Dynamic Source Routing [Jso07]; instead of all nodes broadcasting a message, only nodes that meet the specified properties will. However, since TARP is built in the ad hoc realm, it has no way of preventing an untrusted node from overhearing a message. Also, TARP relies on each node to truthfully inform its neighbors of its capabilities and does not provide a mechanism to validate these claims.

Other ad hoc networking techniques dynamically adjust trust based on observed behavior [He06][LJT04][YPW05]. If the trust is too low, you do not route towards these nodes. These techniques are vulnerable to passive attacks (such as interception). Passive attacks are difficult to observe and are one of the core motivations for our work. These approaches

are also vulnerable to manipulation attacks similar to those on reputation systems. An attacker can operate correctly to build high levels of trust before executing their attack, effectively defeating these techniques.

Sensor networks are very similar to ad hoc networks. Node can easily be compromised and masquerade as valid nodes. False information and resource utilization are some of the principle attacks. Chakrabarti et al. propose a three tiered trust framework for sensor networks [CPR12]. They combine two techniques, dynamic trust values and cryptography. They use cryptography to protect data as it passes through their hierarchy and use dynamic trust values to determine if information should be excluded or included in subsequent calculations. Their trust values are modified based on the accuracy of information provided. Unfortunately, they do not discuss how they determine the accuracy of the information. Further, the reputation manipulation attacks discussed above are still possible.

Finally, Johnson et al. discuss trust in anonymous communications [JSD11]. They propose a model for anonymous communications that incorporates trust of users and nodes. Their model includes the likelihood of node ownership by adversarial nodes, as well as source and destination living in adversarial regions. This model can then be used for route construction to try and pick paths that are highly trusted while not allowing for multi-session de-anonymization (i.e. always using the same nodes leading to a user signature). While their model and analysis is very compelling, it is unclear how well their techniques will scale to that of the Internet. Further, meeting the general goals of avoidance routing, rather than anonymous communications, may be difficult for their model.

8.3 Geographic-Based Routing

One of the principle motivations for avoidance routing is routing on behalf of nation states. Routing that focuses strictly on geopolitical considerations has not yet been widely explored. However, there has been some work on routing that focuses on geographic, rather

than geopolitical, information. Some of the techniques used have been leveraged by avoidance routing.

Several MANET protocols have explored the use of geographic information. Georouting [BMU01][KK00] uses the GPS coordinates of the destination to route from the sender to the receiver. When one wants to send, they look up the last known GPS coordinates for the destination and route that direction. As the packet gets closer to the destination, the coordinates are updated until the destination is reached. Landmark routing [GHP00] is similar, however nodes route towards a landmark which has recently seen the destination rather than GPS coordinates. In either case, the geographic information is used to route to a destination, not around adversarial nodes. Also, we do not believe global GPS or geographic information on routers will be easily available.

Oliviera et al. [OLZ07] explore using geographic information to improve BGP-based routing decisions. They show that by using geographic information encoded in BGP advertisements, one can make better routing decisions while shrinking routing table size. While we do not use our information to make better routing decisions, their approach to augmented BGP advertisements is leveraged by avoidance routing. Francis et al. [Fra94] have also explored using geographical based addresses to help routing decisions and reduce routing tables. Unfortunately, ISPs have generally been against changing routing schemes.

8.4 Routing via Search

Depth-first search has been used before to make routing decisions for both quality-of-service (QoS) and wireless routing. Shin et al. [SCS01] use DFS to find routes that meet certain QoS constraints. Shin et al. use DFS differently than we do, introducing a set of problems that we do not have to deal with. The primary difference is their properties are path-dependent, where our properties are not. This means that how messages arrive at a node is important for this approach in determining the next hop. In their paper, the search is

optimized to solve this problem, while ours is optimized for different purposes.

Stojmenovic et al. [SRV00] use DFS to route in wireless networks. They use GPS and QoS information to try and pick nearby, optimal, next hops. Unfortunately, their system must rediscover routes for every request, since their work is in the MANET area where the routes change quickly. Furthermore, their decisions are based on GPS, which is information we cannot expect to have for Internet routers. Finally, they make decisions based on QoS information which are generally path-dependent. This has the same path-dependent limitation as Shin's system.

In general, many of the MANET routing protocols operate using a search. Because of their mobile nature, it is impossible to build a static graph of the network. To overcome this, protocols such as DSR [Jso07] and AODV [Per03] use searches to find the path. Both protocols send out route request floods to search for the destination. Once the request reaches the destination, it sends a route response based on the path the request used. In this way, the path is found. This basic approach is the same core approach to avoidance routing route construction, except avoidance routing only sends one request rather than a flood. The fundamental problem with both protocols is that they flood in an uncontrolled manner to find any path, making scaling a difficult problem. Further, floods can easily be used as an amplification mechanism for a DoS attack.

8.5 Policy-Based Routing

Avoidance routing is a specific protocol of a set known as policy-based routing [Bra89]. Policy-based routing is any routing scheme where per-packet routing decisions are made based on some specific policy. Current policy-based routing mechanisms [Cis08] generally work on parameters currently carried in the packet, such as source or destination. This allows for the enforcement of blacklists or preference in route selection, but does not provide the dynamics we desire. Our routing decisions are made based on the avoidance criteria

specified by each host.

Avoidance routing may also be viewed as a constraint system, where the criteria are constraints placed on the Internet graph. Younis and Fahmy [YF03] explore constraint-based routing, a combination of policy-based routing and quality-of-service routing. Their goal is find paths through the graph that satisfy a given set of constraints. However, they do not explore avoidance-based policies. Further, they do not solve the problem of how to disseminate or enforce such policies across domains, but only within a single domain (AS).

One policy-based routing protocol of interest is rule-based forwarding (RBF) [PSR09]. In rule-based forwarding, rules are constructed that can treat packets differently. These rules can be based on packet contents or router properties, such as destination or next hop bandwidth. It is unclear how rule-based forwarding could accomplish the goals of avoidance routing without some extensions. However, given these extensions for avoidance routing, a rule-based system could help improve avoidance routing performance. That is, rules could exist to help forwarding of packets given specific criteria. This may be especially true if there is redundancy in avoidance criteria. If many users all wish to avoid China, having a rule regarding China may significantly reduce path construction time.

Another interesting type of routing is predicate routing [RHI03]. Predicate routing views forwarding as a series of predicates. Each packet is treated as a variable in a logical statement. When a packet arrives, it determines which logical predicate it satisfies and forwards based on the predicate. In this way, networks can treat different packets in different ways. This system could work for avoidance routing in a similar way to RBF. The problem with both schemes is they are not dynamic. The rules or predicates must be known prior to the packet arriving. How to change the rules dynamically is an open question. Even if the rules or predicates can be changed on the fly, its not clear how to disseminate security information without using the avoidance routing mechanisms. If the avoidance criteria is kept short or an ISP is only willing to offer a small number of avoidance routes, a predicate routing or RBF type scheme could be used.

Finally, active networks [TSS97] [TW07] provide an interesting possible alternative mechanism for avoidance routing. Active networks allow packets to modify network conditions, such as routing, on-demand. This is accomplished by having packets contain executable code that can be run on routers. To use avoidance routing, a packet would contain code that would dictate to each router how to route it. Unfortunately, this requires the sender to know a lot more information then the current avoidance routing scheme. The sender would need significant topological information. Also, the sender must know how to reconfigure each router. Further, active networks represent a dangerous security risk, as someone could reconfigure the network in a malicious way. Because of this, active networks are not a suitable alternative.

8.6 Multi-Protocol Label Switching

Avoidance routing leverages concepts from multi-protocol label switching (MPLS)[RVC01]. The most important of these is how the avoidance routing forwarding mechanism works. Avoidance routing forwards based on cache IDs, rather than destination address. In general, this is how MPLS forwards as well, using labels rather than destinations. Further, some MPLS mechanisms allow the labels to be changed as packets transit the network, to adjust how these packets transit the network. This is similar to how cache IDs are exchanged from router to router in avoidance routing.

While MPLS is heavily used in the telecom industry, there have been several designs for using it on the Internet. Several of these approaches revolve around traffic engineering [AMA99][LR98]. Traffic engineering, in general, is the manipulation of how traffic transits a network to optimize some utility function. In practice, it can be used to load balance or route around trouble spots. MPLS provides easy mechanisms for achieving this goal. Given knowledge of a graph, one annotates it with characteristics they care about, such as bandwidth. Each path through the graph is determined and given a label. For load

balancing concerns, as a packet enters the network, it is given a label based on the path with the smallest load. This is significantly different than IP routing, as IP path metrics would have to be manipulated on the fly to result in a new routing decision.

Using MPLS for traffic engineering in intra-AS routing [XHB00] and for implementing differentiated services (DiffServ) [FDD02] have both been considered. In general, they both leverage the MPLS mechanism as described above to pick paths that meet there specific goals. Unfortunately, with limited global topological information or cooperation, MPLS as a Internet-scale routing mechanism does not work. Further, while one can determine and label all routes within their AS, doing this on the Internet would lead to the combinatorial problems avodiance routing faces. Instead, MPLS would have to implement a cache system similar to ours in order to meet these goals.

8.7 Consensus

Consensus mechanisms have been widely used throughout computer science. Generally, consensus mechanisms are used for consensus computation [BDM93]. These mechanisms operate under the principal that their are multiple systems calculating and reporting values. How to properly calculate a uniform value, or make a decision based on the values, is determined via consensus computation [OM04]. These systems may be faulty, incorrect or in conflict, and consensus mechanisms are used in order to determine the correct value[Ngu02]. Many of these systems assume faulty, not malicious, nodes [RNB05].

Consensus has been used to detect malicious nodes [PBB07][PBB11]. This is done by quantifying a range of "identifiable" input. Nodes that move outside this range of inputs are believed to be malicious. Many of these mechanisms do not utilize temporal information (volatility) to try and infer malicious nodes. Further, these systems assume that all nodes here all information and that a "liar" lies universally, which we do not.

John et al. propose consensus routing [JKK08], a BGP extension that only makes rout-

ing decisions if their neighbors reach consensus. This works by asking each neighbor their current "view" of the network. They make routing more robust by damping advertisements until all neighbors are in a consistent state. Their results show significant promise. We could possibly extend avoidance routing to wait to change properties until all neighbors are consistent.

Reitblatt et al. use consensus (or consistency) for routing as well [RFR11]. Rather than looking at BGP, their approach looks at software-defined networks (SDN). These networks operate similarly to active networks, how packets traverse the network can be modified easily and quickly through software. Their consistency approach ensures that each packet in a flow can be routed in a consistent manner. Further, one can configure updates such that no changes are made until all switches in a network have achieved a consisted answer.

Finally, consensus approaches to solving the intruder detection problem have also been explored [FB11]. They use the general area of linear logical consensus to determine if an intruder has or has not entered the system. This is accomplished by each system reporting a logical (boolean) value to a central authority (or each other). Using linear logic, an overall truth value can be determined. This value then answers the question, is an intruder present? Avoidance routing uses a similar mechanism, although rather than reporting logical values, we report absolute values. The boolean values are calculated at each node based on these absolute values.

CHAPTER 9

Conclusion

Avoidance routing solves a problem that has generally gone unaddressed on the Internet, how to control who has access to your data as it transits the network. Historically, groups have made great use of intercepted data, and have gone to great lengths to obtain it. While cryptography offers some confidentiality, it does not prevent an adversary from obtaining data or observing it as it goes by. Analyzing traffic patterns has already been shown to provide useful intelligence. Providing user-directed control of Internet routing for security purposes is required.

In this dissertation, we have presented avoidance routing, a mechanism to provide this control. Avoidance routing is a system that allows users to request a route to a destination while specifying criteria of routers they do not trust. Given this criteria, a route to the destination will be found that does not transit routers that have untrusted properties. Several techniques are described that accomplish the goals of avoidance routing.

The first of these techniques discussed is the dissemination of security properties. Security properties are a set of characteristics a router may have, such as geopolitical location or corporate ownership. These properties are disseminated in the "security-vector" of augmented BGP advertisements. We have discussed why we chose to augment BGP and why alternatives were not applicable. Fundamentally, augmenting BGP allows us to interface with existing routers and routing software while allowing us to leverage the information already propagated by the protocol.

Our analysis shows that augmenting BGP with the security-vector is not particularly

costly. Adding a property to the vector is relatively cheap, costing only 16 bytes in our implementation, and could be cheaper depending on the property. As we showed, augmenting a BGP advertisement of 30 hops with 10 total values per hop results in an increase of 1864 bytes, less than 2 KB. This is reasonable considering that the average path length is 4.61 ASes and the maximum observed path is 30 ASes.

The next technique used to allow avoidance routing to operate properly is describes how the disseminated information can be used for routing. This begins with the avoidance request, the message a user sends when they want to be constructed. The avoidance request is a packet sent to the destination that consists of the criteria the user wishes their data to avoid. This request will traverse the network in a depth-first manner, searching for a path that is valid.

When a router receives an avoidance request, it begins by determining if it already knows how to service this request by data stored in its caches. If the router does not have this information, it begins a search for a valid path. It will search all of its known advertisements to the destination, determining if any of them satisfy the given criteria. If so, it terminates the search and immediately sends the request along. If not, it will order its neighbors by a heuristic and forward the message to the neighbor at the top of the order. The packet may reach a dead-end, a node who cannot forward the message. In this case, this router will report failure to the previous router, who will then try the next neighbor in its order.

If the request reaches the destination, a success message is returned along the constructed path. Each node on the path will create a cache entry based on the success message and then forward the message back along the reverse path. It is possible that routing policy prevents the reverse path to be used. In this case, the receiver will begin a new request to construct the reverse path. The sender will receive the reverse path request (identical to the forward path request) and send success. Further, sender and receiver may have different criteria. These criteria will be unioned (if not in conflict) when the reverse

path request is sent out.

As we described, forwarding actually occurs using an MPLS-style mechanism. Avoidance routing uses caches, with cache labels, to identify the forwarding path. When a user actually sends data to be avoidance routed, it will add the cache label to the packet. Thus, when a router receives a packet to be avoidance routed, it looks up the forwarding interface based on label, rather than IP address.

Our extensive experiments have shown that avoidance routing performs quite well. We show that avoidance routing tends to find paths that are, on average, only 2.5 hops longer than the standard routes when using a uniform property distribution and standard BGP policies. Further, when considering a geopolitical property distribution, the average increase is less than 12 hops. While some paths may be extremely poor (on the order of 200 additional hops), the vast majority of paths we found had and increase of less than 5 hops.

Our depth-first mechanism can have the tendency to visit extra nodes that do not contribute to the final path. Our results confirm this property of DFS. Both geopolitical and uniform distributions may visit 50 additional nodes, on average. Again, in some cases, this may be exceptionally poor, visiting 2500 extraneous nodes. However, like with path length, 90% of paths visit less than 10 additional hops, although this varies based on the number of nodes removed by the selected properties. Further, our results show that the majority of routes discovered are able to find this route within the first two hops. That is, one of the first two routers has a completely valid advertisement to the destination, foregoing the entire search phase and saving resources on nodes.

A final important result regarding avoidance routing path construction is that packet forwarding is unaffected by avoidance routing. Our implementation and results confirm that non-avoidance routed packets (packets using standard routing) suffer no performance penalty when using avoidance routing-enabled routers. Further, once a path is constructed, packets forwarded by avoidance routing suffer no penalty.

Avoidance routing also provides a partial disclosure option. This option, neighbor-only avoidance routing, trades performance for privacy. In this case, each neighbor is only aware of the properties of their neighbors. Thus, a node cannot know if path is completely valid and must always run a full search. Because of this, neighbor-only avoidance routing performs worse than standard avoidance routing.

While the avoidance routing path construction and forwarding mechanisms have positive results, they rely heavily on the information in the security vector. How we protect this information is clearly important. Our discussion on the avoidance routing consensus mechanism covers how this information is protected. By having each node determine how strong the agreement on each property is, as well as the quality of each neighbor as a "voter", we can determine how well to trust the given information. While other approaches, such as authentication, may provide stronger guarantees, we clearly describe why these approaches are not applicable for avoidance routing.

Our analysis of the avoidance routing consensus mechanism shows that it can perform quite strongly as a protection mechanism. In general, an adversary requires control over a significant number of neighbors to invalidate the properties of a neighbor. Further, avoidance routing will never use its consensus mechanism to override the beliefs a node. Thus, no matter how many neighbors are controlled by an adversary, a node will not forward to a neighbor it feels is invalid. A user will specify how strong beliefs must be to use a given node. A user may be extremely paranoid, wanting high levels of agreement for the rare case that "everyone got it wrong".

The discussion in this dissertation has shown that avoidance routing has great potential. Future techniques described, such as partial maps, may alleviate some of the worst-case scenarios and increase overall performance. Other possibilities, such as receiver-oriented avoidance routing, may open up whole new areas of research.

Regardless of possible future opportunities, the results shown in this dissertation clearly state that avoidance routing is a viable routing mechanism. While avoidance routing may

sometimes perform poorly, the majority of the time, it performs very well. Further, avoidance routing provides users the control they require in order to ensure that their data does not transit untrusted nodes. Without avoidance routing, this control is unavailable, providing users no way to prevent adversaries from observing, copying or poisoning their data. With the techniques described, and their performance characteristics, we have made this capability available to users and contributed to the security of routing. We encourage other researches to continue from this starting point, creating more techniques that may perform better and provide even stronger security capabilities.

References

[Aka08] "Akamai." http://www.akamai.com, 2008.

[AKB06] L. Abusalah, A. Khokhar, G. BenBrahim, and W. ElHajj. "TARP: Trust-Aware Routing Protocol." In *International Wireless Communications and Mobile Computing Conference*, 2006.

[AMA99] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. "Requirements for Traffic Engineering Over MPLS." RFC 2702, 1999.

[And96] R. Anderson. "Hiding Routing Information." *Information Hiding*, **1174**:137–150, 1996.

[AS07] B. Awerbuch and C. Scheideler. "Toward Scalable and Robust Overlay Networks." In *International Workshop on Peer-To-Peer Systems*, 2007.

[ATL10] A. Afanasyev, N. Tilley, B. Longstaff, and L. Zhang. "BGP Routing Table: Trends and Challenges." In *Proceedings of the 12th Youth Technological Conference "High Technologies and Intellectual Systems"*, Moscow, Russia, April 2010.

[BBC08] "BBC News. Sweden approves wiretapping law." http://news.bbc.co.uk/2/hi/europe/7463333.stm, June 19 2008.

[BBK02] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. "Scalable Application Layer Multicast." In *ACM SIGCOMM*, 2002.

[BDM93] M. Barborak, A. Dahbura, and M. Malek. "The Consensus Problem in Fault-Tolerant Computing." *ACM Computing Surveys*, **25**(2):171–220, June 1993.

[Bei07] I. Beijnum. "Old IPv4 flaws resurface with IPv6." http://arstechnica.com/gadgets/2007/05/old-ipv4-flaws-resurface-with-ipv6/, 2007. Ars Technica.

[Bil12] Nick Bilton. "Girls Around Me: An App Takes Creepy to a New Level." http://bits.blogs.nytimes.com/2012/03/30/girls-around-me-ios-app-akes-creepy-to-a-new-level/, 2012. The New York Time Bits Blog.

[BMU01] I. S. P. Bose, P. Morin, and J. Urrutia. "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks." In *ACM/Kluwer Wireless Networks*, 2001.

[Bra89] H-W. Braun. "Models of Policy Based Routing." RFC 1104, 1989.

[Cai12] "Caida AS Relationships." http://www.caida.org/data/active/as-relationships/index.xml, 2012.

[CHM02] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. "Protecting Freedom of Information Online with Freenet." *IEEE Internet Computing*, **6**(1):40–49, January-February 2002.

[Cis08] "Configuring Policy-Based Routing." http://www.cisco.com/en/US/docs/ios/12_0/qos/configuration/guide/qcpolicy.html, 2008.

[Cis12a] "Cisco CRS-3 Label Switch Processor." http://www.cisco.com/en/US/prod/collateral/routers/ps5763/data _sheet_c78-659947.html, 2012.

[Cis12b] "Cisco CRS Route Processor Data Sheet." http://www.cisco.com/en/US/prod/collateral/routers/ps5763/data _sheet_c78-659773.html, 2012.

[Cor08] Microsoft Corporation. "Source Address Spoofing." http://technet.microsoft.com/en-us/library/cc723706.aspx, 2008.

[CPR12] A. Chakrabarti, V. Parekh, and A. Ruia. "A Trust Based Routing Scheme for Wireless Sensor Networks." In *Advances in Computer Science and Information Technology. Networks and Communications*, volume 84 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 159–169. Springer Berlin Heidelberg, 2012.

[CRB03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. "Making Gnutella-like P2P Systems Scalable." In *Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003.

[CRZ00] Y. Chu, S. Rao, and H. Zhang. "A Case for End System Multicast." In *ACM SIGMETRICS*, 2000.

[Cym12a] "Team Cymru BGP/ASN Analysis Report." http://www.cymru.com/BGP/summary.html, 2012.

[Cym12b] "Team Cymru IP to ASN Mapping." http://www.team-cymru.org/ Services/ip-to-asn.html, 2012.

[Dai04] L. Daigle. "WHOIS Protocol Specification." RFC 3912, 2004.

[DH98] S. Deering and R. Hinden. "Internet Protocol, Version 6 (IPv6)." RFC 2460, 1998.

[Dij59] E. W. Dijkstra. "A note on two problems of connexion with graphs." *Numerische Mathematik*, **1**, 1959.

[DKF07] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, Kc Claffy, and G. Riley. "AS Relationships: Inference and Validation." *ACM SIGCOMM Computer Communication Review*, **37**(1):22–40, 2007.

[FB11] A. Fagiolini and A. Bicchi. "Logical Consensus for Distributed and Robust Intrusion Detection." *CoRR*, 2011.

[FDD02] F. Le Faucheur, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen. "Multi-Protocol Label Switching (MPLS) Support for Differentiated Services." RFC 3270, 2002.

[FFF99] M. Faloutsos, P. Faloutsos, and C. Faloutsos. "On Power-law Relationships of the Internet Topology." *SIGCOMM Comput. Commun. Rev.*, **29**(4):251–262, August 1999.

[Fra94] P. Francis. "Comparison of Geographical and Provider-rooted Internet Addressing." *Computer Networks and ISDN Systems*, **27**(6), 1994.

[Fre12] "Free printable maps of Europe." http://www.freeworldmaps.net/printable/europe/, 2012.

[Gao01] L. Gao. "On Inferring Autonomous System Relationship in the Internet." In *IEEE/ACM Transactions on Networking*, December 2001.

[Gar00] L. Garber. "Denial-of-Service Attacks Rip the Internet." *IEEE Computer*, pp. 12–17, April 2000.

[GBH09] C. Gates, M. Bishop, and J. Hunker. "Sisterhood of the Travelling Packets." In *NSPW*, 2009.

[GHP00] M. Gerla, X. Hong, and G. Pei. "Landmark Routing for Large Ad Hoc Wireless Networks." In *IEEE Globecom 2000*, 2000.

[Gnu08] "Gnutella." http://www.gnutella.com, 2008.

[Gor11] S. Gorman. "China Hackers Hit U.S. Chamber." http://online.wsj.com/article/SB10001424052970204058404577110541568535300.html, December 2011.

[GR02] R. Govindan and P. Radoslavov. "An Analysis of The Internal Structure of Large Autonomous Systems." Technical Report Technical Report 02-777, University of Southern California, 2002.

[GT00] R. Govindan and H Tangmunarunkit. "Heuristics for Internet Map Discovery." In *IEEE Infocom 2000*, 2000.

[Gut02] P. Gutman. "Lessons Learned in Implementing and Deploying Crypto Software." In *Usenix Security Symposium*, 2002.

[He06] L. He. "A Novel Scheme on Building a Trusted IP Routing Infrastructure." In *International Conference on Networking and Services*, 2006.

[HerBC] Herodotus. "Histories of Herodotus.", 557-479 BC.

[Hes00] R. Hesketh. *Fortitude: The D-Day Deception Campaign.* The Overlook Press, 2000.

[Hoe09] D. Hoelzer. "The Dangers of Source Routing." http://enclave forensics.com/Blog/files/dbe04629c14a2d07495a38bbf2fc98d9-5.html, 2009.

[HR] S. Huffman and M. Reifer. "Method for geolocating logical network addresses." United States Patent 6,947,978. Filed December, 2000. Issued September 2005.

[HZN09] K. Hoffman, D. Zage, and C. Nita-Rotaru. "A Survey of Attack and Defense Techniques for Reputation Systems." *ACM Comput. Surv.*, **42**(1), December 2009.

[ISI02] "Intermediate System to Intermediate System." ISO/IEC 10589:2002, 2002.

[JFS10] A. M. Johnson, J. Feigenbaum, and P. Syverson. "Preventing Active Timing Attacks in Low-Latency Anonymous Communication." In *Privacy Enhancing Technologies Symposium (PETS)*, 2010.

[JKK08] J.P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkatarmani. "Consensus Routing: The Internet as a Distributed System." In *NSDI '08*, 2008.

[JSD11] A. M. Johnson, P. Syverson, R. Dingledine, and N. Mathewson. "Trust-Based Anonymous Communication: Adversary Models and Routing Algorithms." In *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[Jso07] D. Jsohnson. "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4." RFC 4728, 2007.

[Jun12] Juniper. "T Series Core Routers." http://www.juniper.net/us/en/local/pdf/datasheets/1000051-en.pdf, 2012.

[Jus08] US Department of Justice memo. "Chinese National Sentenced For Committing Economic Espionage With the Intent to Benefit China Navy Research Center." http://www.cybercrime.gov/mengSent.pdf, June 2008.

[Kah67] D. Kahn. *The Codebreakers.* MacMillan, 1967.

[KC09] R. Kerr and R. Cohen. "Smart Cheaters do Prosper: Defeating Trust and Reputation Systems." In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, 2009.

[KGA07] S. Khurana, N. Gupta, and N. Aneja. "Minimum Exposed Path to the Attack (MEPA) in Mobile Ad hoc Network (MANET)." In *International Conference on Networking*, 2007.

[KJK06] E. Katz-Bassett, J.P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. "Towards IP Geolocation using Delay and Topology Measurements." In *ACM SIGCOMM Conference on Internet Measurement*, 2006.

[KK00] B. Karp and H. T. Kung. "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks." In *ACM MobiCom 00*, 2000.

[KLS00] S. Kent, C. Lynn, and K. Seo. "Secure Border Gateway Protocol (Secure BGP)." *IEEE Journal on Selected Areas in Communication*, **18**(4), April 2000.

[KMC00] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F. Kasshoek. "The Click Modular Router." In *ACM Trans. on Computer Systems (TOCS)*, 2000.

[Kor85] R. Korf. "Depth-First Iterative-Deepening: An Optimal Admissible Tree Search." *Artificial Intelligence*, **27**(1):97–109, September 1985.

[Kor97] R. Korf. "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases." In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, AAAI'97/IAAI'97, 1997.

[Lew08] J. A. Lewis. "Securing Cyberspace for the 44th President." Center for Strategic and International Studies, 2008.

[LGL96] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. "SOCKS Protocol Version 5." RFC 1928, 1996.

[LJC08] P. Laskowski, B. Johnson, and J. Chuang. "User-Directed Routing: From Theory, towards Practice." In *International Workshop on Economics of Networked Systems*, 2008.

[LJT04] Z. Liu, A. Joy, and R. Thompson. "A Dynamic Trust Model for Mobile Ad Hoc Networks." In *IEEE International Workshop on Future Trends of Distributed Computing Systems*, 2004.

[LPR04] J. Li, G. Popek, and P. Reiher. "Resilient Self-Organizing Overlay Networks for Security Update Delivery." *IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks*, **22**(1), January 2004.

[LR98] T. Li and Y. Rekhter. "Provider Architecture for Differentiated Services and Traffic Engineering (PASTE)." RFC 2430, 1998.

[MCA09] T. Moore, R. Clayton, and R. Anderson. "The Economics of Online Crime." *The Journal of Economic Perspectives*, **23**(3):3–20, 2009.

[MD05] S. Murdoch and G. Danezis. "Low-Cost Traffic Analysis of Tor." In *IEEE Symposium on Security and Privacy*, 2005.

[MG06] S. Marti and H. Garcia-Molina. "Taxonomy of Trust: Categorizing P2P Reputation Systems." *Computer Networks*, **50**(4):472 – 484, 2006.

[MK11] J. Mirkovic and E. Kissel. "Comparative Evaluation of Spoofind Defenses." *IEEE Transactions on Dependable and Secure Computing*, **8**(2):218–232, March - April 2011.

[MO09] J. A. Muir and P. C. Van Oorschot. "Internet Geolocation: Evasion and Counterevasion." *ACM Computer Surveys*, **42**(1), December 2009.

[Moy98] J. Moy. "OSPF Version 2." RFC 2328, 1998.

[Ngu02] N.T. Nguyen. "Consensus System for Solving Conflicts in Distributed Systems." *Information SciencesInformatics and Computer Science*, **147**(1-4):91–122, October 2002.

[NZ02] T.S.E. Ng and H. Zhang. "Predicting Internet Network Distance with Coordinates-Based Approaches." In *INFOCOM*, 2002.

[OLZ07] R. Oliveira, M. Lad, B. Zhang, and L. Zhang. "Geographically Informed Inter-Domain Routing." In *IEEE ICNP*, 2007.

[OM04] R. Olfati-Saber and R.M. Murray. "Consensus Problems in Networks of Agents with Switching Topology and Time-delays." *IEEE Transactions on Automatic Control*, **49**(9):1520 – 1533, September 2004.

[OPW08] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. "In Search of the Elusive Ground Truth: The Internet's AS-Level Connectivity Structure." In *ACM SIGMETRICS*, 2008.

[OPW10] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. "The (in)Completeness of the Observed Internet AS-level Structure." In *IEEE/ACM Transactions on Networking*, 2010.

[PAA07] "S. 1927 – 110th Congress: Protect America Act of 2007." http://www.govtrack.us/congress/billxpd?bill=s110-1927, 2007.

[PBB07] F. Pasqualetti, A. Bicchi, and F. Bullo. "Distributed intrusion detection for secure consensus computations." In *Proc. IEEE Int. Conf. on Decision and Control*, 2007.

[PBB11] F. Pasqualetti, A. Bicchi, and F. Bullo. "Consensus Computation in Unreliable Networks: A System Theoretic Approach." *IEEE Transactions on Automatic Control*, **57**(1), 2011.

[Per03] C. Perkins. "Ad hoc On-Demand Distance Vector (AODV) Routing." RFC 3561, 2003.

[Pos81] J. Postel. "Internet Protocol." RFC 791, 1981.

[Pri05] G. Price. "PKI Challenges: An Industry Analysis." In *International Workshop on Applied Public Key Infrastructure: IWAP*, 2005.

[PSR09] L. Popa, I. Stoica, and S Ratnasamy. "Rule-based Forwarding (RBF): improving the Internet's flexibility and security." In *HotNets VIII*, 2009.

[Qua12] "Quagga Software Routing Suite." http://www.nongnu.org/quagga/, 2012.

[RD09] R. Rohozinski and R. Deibert, editors. *Tracking GhostNet: Investigating a Cyber Espionage Network*. Information Warfare Monitor, 2009.

[RFR11] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. "Consistent Updates for Software-Defined Networks: Change you can Believe in!" In *ACM Workshop on Hot Topics in Networks*, 2011.

[RHI03] T. Roscoe, S. Hand, R. Isaacs, R. Mortier, and P. Jardetzky. "Predicate routing: enabling controlled networking." *SIGCOMM Comput. Commun. Rev.*, **33**(1), 2003.

[RLH06] Y. Rekhter, T. Li, and S. Hares. "A Border Gateway Protocol 4." RFC 4271, 2006.

[RNB05] M.G. Rabbat, R.D. Nowak, and J.A. Bucklew. "Generalized consensus computation in networked systems with erasure links." In *Workshop on Signal Processing Advances in Wireless Communications*, 2005.

[RSA12] "What are the cryptographic policies of some countries?" http://www.rsa.com/rsalabs/node.asp?id=2333, 2012. RSA Laboratories.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. "Multiprotocol Label Switching Architecture." RFC 3031, 2001.

[SAR02] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. "Characterizing the Internet Hierarchy from Multiple Vantage Points." In *IEEE INFOCOM*, 2002.

[Sch96] B. Schneier. *Applied Cryptography*, pp. 10–13. John Wiley and Sons, Inc., 1996.

[Sch00] B. Schneier. *Secrets and Lies*, pp. 34–35. John Wiley and Sons, Inc., 2000.

[Sch08] F. Schneider. "Network Neutrality versus Network Trustworthiness?" *IEEE Security and Privacy*, **6**(4), July-August 2008.

[SCS01] D. Shin, E. K. P. Chong, and H. Siegel. "A Multiconstraint QoS Routing Scheme using the Depth-First Search Method with Limited Crankbacks." In *IEEE Workshop on High Performance Switching and Routing*, 2001.

[Sin00] S. Singh. *The Code Book*. Anchor Books, 2000.

[SRV00] I. Stojmenovic, M. Russel, and B. Vukojevic. "Depth First Search and Location Based Localized Routing and QoS Routing in Wireless Networks." In *Computers and Informatics*, 2000.

[SSK98] R. Siamwalla, R. Sharma, and S. Keshav. "Discovering Internet Topology." http://www.cs.cornell.edu/skeshav/papers/discovery.pdf, 1998.

[SSW10] Y. Schwartz, Y. Shavitt, and U. Weinsberg. "A Measurement Study of the Origins of End-to-End Delay Variations." In *Passive and Active Measurement*, 2010.

[Su10] M.Y. Su. "WARP: A Wormhole-Avoidance Routing Protocol by Anamoly Detection in Mobile Ad hoc Networks." *Computer Security*, **29**, March 2010.

[Sue21] Suetonius. "Life of Julius Caesar.", 121.

[Tor12] "Tor Project." http://www.torproject.org, 2012.

[TSS97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. "A Survey of Active Network Research." In *IEEE Commun.*, 1997.

[Tuc66] B. Tuchman. *The Zimmerman Telegram*. MacMillan, 1966.

[TW07] D. L. Tennenhouse and D. J. Wetherall. "Towards an Active Network Architecture." *SIGCOMM Comput. Commun. Rev.*, **37**(5), 2007.

[Ucl12] "Internet Topology." http://irl.cs.ucla.edu/topology/, 2012.

[WCJ05] X. Wang, S. Chen, and S. Jajodia. "Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet." In *ACM Conference on Computer Communications Security (CCS)*, 2005.

[WMS11] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose. "Phonotactic Reconstruction of Encrypted VoIP Conversations." In *IEEE Symposium on Security and Privacy*, 2011.

[Woo12] P. Wood. "Internet Security Threat Report." http://www.symantec.com/threatreport/, 2012. Symantec Corporation.

[WZM07] J. Wu, Y. Zhang, Z. M. Mao, and K. G. Shin. "Internet Routing Resilience to Failures: Analysis and Implications." In *ACM CoNEXT*, 2007.

[XHB00] X. Xiao, A. Hannan, B. Bailey, and L.M. Ni. "Traffic Engineering with MPLS in the Internet." *Network, IEEE*, **14**(2):28 – 33, March - April 2000.

[YF03] O. Younis and S. Fahmy. "Constraint-Based Routing in the Internet: Basic Principles and Recent Research." In *IEEE Communications Surveys and Tutorial*, 2003.

[YG02] B. Yang and H. Garcia-Molina. "Improving Search in Peer-to-Peer Networks." In *International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[YNK01] S. Yi, P. Naldurg, and R. Kravets. "A Security-Aware Routing Protocol for Wireless Ad Hoc Networks." In *ACM Symposium on Mobile Ad Hoc Networking & Computing*, 2001.

[YPW05] R. Yang, Q. Pan, W. Wang, and M. Li. "Secure Enhancement Scheme for Routing Protocol in Mobile Ad Hoc Networks." In *IEEE International Conference on Distributed Computing Systems Workshops*, 2005.

[ZFG04] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. "On Flow Correlation Attacks and Countermeasures in Mix Networks." In *Privacy Enhancing Technologies*, 2004.

[ZHH11] X. Zhang, H. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. "SCION: Scalability, Control, and Isolation On Next-Generation Networks." In *IEEE Symposium on Security and Privacy*, 2011.

[ZL07] H. Zlatokrilov and H. Levy. "Navigation in Distance Vector Spaces and Its Use for Node Avoidance Routing." In *IEEE Infocom 2007*, 2007.

[ZL08] H. Zlatokrilov and H. Levy. "Area Avoidance Routing in Distance Vector Networks." In *IEEE Infocom 2008*, 2008.

[ZYF10] N. Zhang, W. Yu, X. Fu, and S.K. Das. "gPath: A Game-Theoretic Path Selection Algorithm to Protect Tor's Anonymity." In *Proceedings of the First International Conference on Decision and Game Theory for Security*, GameSec'10, 2010.

[ZZT05] X. Zhao, B. Zhang, A. Terzis, D. Massey, and L. Zhang. "The Impact of Link Failure Location on Routing Dynamics: A Formal Analysis." In *ACM SIGCOMM Asia Workshop*, 2005.