Adversarial Degradation of the Availability of Routing Infrastructures and Other Internet-Scale Distributed Systems

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Maxfield Joseph Schuchard

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Nicholas Hopper

June, 2016

Acknowledgements

There are an exceptionally large number of people without whom this work would never have happened. Ironically, first and foremost are two of the most computer illiterate people I know, my parents. Mom and Dad, this work is yours as much as it is mine. On the opposite end of the spectrum are my "academic parents": Nick Hopper, Yongdae Kim, John Riedl, and Marda Olson. I do not know where my life would have taken me without your influence in it, but I am confident it would not have been anywhere near as enjoyable and fulfilling.

I would like to thank all of my collaborators and co-authors in no particular order: Eugene Vasserman, Aziz Mohaisen, John Geddes, Christopher Thompson, Mike Schliep, Denis Foo Kune, Rob Jansen, Victor Heorhiadi, Alex Dean, Connor Van Drisse, and Stephen Mylabathula. No quality research happens in a vacuum, and this work is no exception. I know I am not the easiest person to work with (an understatement if ever there was one), you all have the patience of saints and I am infinitely thankful for that.

I also need to thank my friends: Corey Spaith, Dave Ankarlo, Matthew Root, Jorel Rohan, Jacob Beyer, Wilson Knorr, Scott Miller, Richard Thomas, and Leonie Mostert. Graduate school is an excruciating trial at best, and mine was personally challenging above and beyond for reasons all of you are far too familiar with. I can honestly say would not be still standing without you guys, and for that you have my undying gratitude.

Last but not least I would like to thank the other members of my committee, Stephan McCamant and Andrew Odlyzko, who volunteered their time and expertise.

I would also like to acknowledge both the National Science Foundation and the University of Minnesota's Doctoral Dissertation Fellowship for funding this work.

Dedication

To that little voice inside of all of us that ask, *"I know you shouldn't be able to do that, but what would happen if I did. . . "*

Abstract

The Internet relies on its routing infrastructure, a globally spanning distributed system of special purpose computers call routers, to deliver packets between hosts. In order to build the paths data will travel, routers execute a routing protocol called the Border Gateway Protocol, or BGP. BGP is built to be decentralized and highly accommodating to arbitrary preferences of the organizations that own routers. This dissertation focuses on examining the following thesis statement. *The current state of BGP, coupled with the Internet's extreme level of topological complexity, allows adversaries who can interact with BGP routers to degrade the availability properties of both the entire Internet routing infrastructure and other Internet-scale distributed systems.*

The research in this work breaks down into two independent arcs. The first arc focuses on attacks which aim to disrupt the availability of large portions on the Internet's routing infrastructure. Through both simulation and experimentation with representative devices, this work demonstrates that a variety of adversaries can prevent large portions of the Internet from being able to correctly build paths to end destinations. The second arc focuses on how those who control routers, and therefore can decide how the routers will select paths, can attack the availability of distributed systems which closely interact with the transit infrastructure of the Internet. Specifically the work shows how, by altering the BGP decision making process slightly, a variety of systems, ranging from censorship circumvention tools to surveillance systems, can be defeated by such an adversary without loss of general connectivity.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet, at its most basic level, is a collection of independently owned and administered networks. These independent networks are termed *autonomous systems*, or ASes. ASes connect themselves to each other in order to facilitate communication between hosts that reside in two different ASes. However, each AS does not have a direct connection to every other AS, rather ASes will possess a limited number of connections to other ASes. In the majority of instances, in order for data to reach a destination AS which its source AS is not directly connected to, it will have to traverse a path consisting of multiple ASes.

This arrangement presents a collection of challenges to autonomous systems. How does knowledge of what links exist get shared? How does that knowledge get converted into paths between ASes? What happens when multiple paths exist, which should be used? In order to answer these and other questions, ASes deploy and operate special purpose computers called *routers*. Routers are charged with two main tasks, building paths to end destinations, and forwarding data along the best known path towards its destination.

Routers connect to other routers over direct links and run a *routing protocol*, which they use to exchange network reachability information. Routers also rely on their routing protocol to select the current best path out of all currently valid routes to a given destination. The routing protocol spoken by routers connecting together different ASes is the *Border Gateway Protocol*, or simply BGP. We can actually picture the Internet as a single, globally spanning, distributed system comprised of routers executing BGP, this system is commonly referred to as the Internet's *routing infrastructure*.

BGP has existed in its current form since 1994, with the most recent update occurring in 2006. Since that update the only changes to BGP have come in the form of optional "bolt on" extensions. While there has been little change to the BGP protocols since 1994, the AS topology has changed radically. We can see the scale of this by examining the data aggregated at [1]. For example, the raw size of the Internet in terms of ASes that are actively advertising prefixes has increased 16.5 fold between when BGP was released and now. The same goes for the raw number of IP prefixes that are advertised to the Internet has increased 37 fold. Since the last update to BGP in 2006 the average number of unique paths for each destination that an AS knows has increase from 1.997 to 2.942.

This growing separation between the topology BGP was first deployed to and the one it is expected to function in today is exacerbated by under-provisioned routing hardware. Because of the high performance hardware required for routers to accomplish their packet switching tasks, hardware routers have high price tags. The result of these high costs is both an over-subscription of router resources and a noticeable difference between the computational capacity of commodity hardware and the route processing components of routers. Several times each year accidental misconfigurations of routers highlight this lack of computational "head room" coupled with BGP's native weaknesses by causing router failures which are felt by end users around the world [2, 3].

These incidents raise a troubling question. If the mismatch between design and deployment environments, along with accidental misconfigurations, can cause large disruptions in the inter-AS routing infrastructure, what could an adversary achieve? On the other hand, what do ASes gain from this increase in topological richness? With the increase in path diversity, what new capabilities do large ASes and nation states possess?

1.1 Thesis Statement

The central thesis of this dissertation is as follows:

The current state of BGP, coupled with the Internet's extreme level of topological complexity, allows adversaries who can interact with BGP routers to degrade the availability properties of both the entire Internet routing infrastructure and other Internet-scale distributed systems.

My research supporting this thesis breaks down into two main arcs. The first arc focuses on attacks which target the availability of the Internet's routing infrastructure holistically. Rather than simply preventing a single BGP router from functioning correctly, the adversary in question has the goal of preventing *all* BGP routers from being able to route to *all* destinations. The second arc examines how autonomous systems can use their ability to make arbitrary routing decisions in order to undermine the availability of Internet scale distributed systems *while still correctly providing connectivity to their hosts.*

1.2 Outline

This dissertation will explore each of these arcs of research independently. In addition, the experimental framework used to evaluate the attacks will also be covered. The outline of this dissertation, along with major contributions of each arc of research are as follows.

Simulating the Internet Routing Infrastructure (Chapter 3)

Evaluating the efficacy of attacks either launched against or launched by large portions of the Internet routing infrastructure is a complex research challenge itself. For ethical reasons we can not directly test these attacks in the wild. In this chapter we will examine how simulators which accurately capture the dynamics of the systems we are interested in were built. Additionally we will talk about how scalability issues were handled when trying to simulate the entirety of the Internet's routing infrastructure. The primary contributions of this Chapter are the Stormcaller simulator in Section 3.2 and the Nightwing simulator in Section 3.3. The Stormcaller [4] simulator was built to model the dynamics of large systems of BGP routers over time. The goal of Stormcaller is to correctly model how each router's memory usage, processing load, and pending routing protocol message queues evolves over a given scenario. Nightwing [5] on the other hand only concerns itself with what each router's path selection decisions are given a particular collection of routing policies, and ignores low level details about a router's state.

Disrupting Systems of BGP Routers (Chapters 4 and 5)

These two chapters explore how a collection of different adversaries can undermine the ability of victim routers to do their most fundamental job, making path selection decisions. The adversaries covered in this work are wide ranging in capabilities. In Chapter 4 we will consider unprivileged adversaries who are not in direct control of any BGP routers. Chapter 5 examines more powerful adversaries who directly control BGP routers.

The goal of both categories of adversary are similar, they wish to prevent victim routers from making correct routing decisions in a timely manner, however because of their radically disparate capabilities, the actions taken by each type of adversary are quite different. Our unprivileged adversary in Chapter 4 will exploit BGP's poor

response to link failures. Specifically the adversary will repeatedly trigger link failures at key points in the Internet's topology, which will cause nearly all routers that make up the topology to flood each other with routing protocol messages, creating computational backlog which prevents the routing infrastructure from reaching a functional state again. The adversary in control of BGP routers in Chapter 5 will exploit both assumptions made by the developers who have built the BGP implementations used by routers and mismatches between BGP's formal specification and actually deployed implementations. We will demonstrate how our adversary can build BGP messages that are fully protocol compliant, but will result in a router crashing when it attempts to process them.

Routing Capable Adversaries (Chapters 6 and 7)

While the prior chapters focus on attacking systems of BGP routers, in Chapters 6 and 7 we will focus on how routers can act as an adversary to other distributed systems. We will examine what we term *routing capable adversaries*, entities that can strategically make routing decisions in an effort to undermine the security properties of other distributed systems. Moreover we will show that our routing capable adversary can accomplish this goal *without* negatively impacting its ability to successfully deliver traffic to and from its AS. Both chapters will focus on examining how routing capable adversaries can attack the availability of a collection of distributed systems which require privileged position in the core of the Internet in order to function. In Chapter 6 we will focus on a particular example of such a system, *decoy routing*. Decoy routing is a form of "end to middle" censorship circumvention system that is located in the core of the Internet rather than being run from end hosts. We will demonstrate that by simple alterations to how BGP selects its best path, a routing capable adversary can prevent users inside its AS from accessing a decoy routing system with essentially no loss in end destination connectivity. In Chapter 7 we will consider how routing decisions are also economic decisions. This in turn means that our routing capable adversary is a powerful economic adversary as well. Given this capability, our adversary has an unconventional way to attack the availability of a distributed system, impose an economic cost on any AS that deploys it. Incentivizing the hosting AS to remove any nodes belong to the distributed system.

Chapter 2

Background

2.1 The Internet as a Graph

The Internet is not a single monolithic network, rather it is composed of multiple networks called autonomous systems, or *ASes*. An AS is defined as a collection of end hosts managed by a single administrative entity. We can model this system using a graph $G = (V, E)$, as shown in Figure 2.1, where V represents the set of autonomous systems and E is the physical inter-connectivity. Two vertices $v_i, v_j \in V$ that represent two ASes, say AS1 and AS2, have an edge $e_{ij} \in E$ between them if and only if AS1 and AS2 are physically connected. The degree of an AS is defined as the number of edges incident to that ASes' vertex representation in G. For example, the degrees of AS5, AS7, AS3, AS4, and AS1 in Figure 2.1 are $1, 2, 3, 4,$ and 5, respectively. As is hinted at by this figure, ASes have a wide amount of variance in terms of AS level connectivity. Some ASes have very high degree; these ASes are considered *core* ASes. Other ASes have very low degrees of connectivity, sitting at the outskirts of the Internet; these are *fringe* ASes. An AS is referred to as a *stub* if it is connected to the Internet graph via a single edge. Putting it another way, all ASes with degree 1 are stubs. (In Figure 2.1, AS5 is a stub.)

Since the underlying physical infrastructure does not allow for complete pairwise connectivity between networks, the network traffic traveling between any two ASes may need to go through a number of intermediate ASes who are physically connected to each other. ASes which agree to forward network traffic toward its end destination for adjacent ASes are referred to as *transit* ASes. With the exception of stub ASes, which only have one adjacent AS to send data to and receive it from, ASes will generally have multiple different paths which could be taken by their data as it traverses the AS level graph. The task of building, preferentially ordering, and using these paths falls to specialized hosts called inter-AS *routers*.

2.2 Routers

Routers are network hosts tasked with building paths to end destinations in layer three networks, most notably the Internet. Routers fall into two broad categories, *intra-AS routers* which handle the delivery of traffic between hosts located inside the same AS, and *inter-AS routers* which handle finding paths between hosts in different ASes. In this

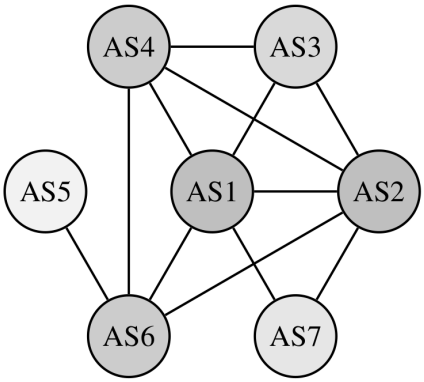


Figure 2.1: An illustration of the Internet as a graph with edges representing intra-AS connectivity.

dissertation we will focus exclusively on the latter. In order to accomplish their task, routers exchange reachability information with other routers using a routing protocol. Routers are often, but not always, responsible for forwarding data plane traffic using the paths they have built. Routers can be broadly partitioned into two categories: hardware routers and software routers.

A hardware router is made up of three components: a collection of line cards, a switching fabric, and a general purpose route processor. Line cards combine a network interface, high performance memory, and a special purpose packet processor in order to receive, make forwarding decision on, and transmit data plane traffic. Line cards are linked together via a high performance switching fabric, designed to move large numbers of packets quickly from ingress to egress line cards. Lastly, routers contain a route processor and general purpose memory. It is the route processor that builds and maintains a routing information base containing possible paths to destinations. The route processor also does best path selection and installs winning routes into forwarding tables stored in high performance memory of line cards. The route processor is also increasingly called upon to provide support for functionality beyond routing as well. For example, route processors can be used to support Multi Label Packet Switching and VPN setups in addition to routing duties.

Hardware routers are constructed using high-performance, highly specialized components in order to cope with the task of forwarding millions to billions of packets per second. The downside of hardware routers can be summed up in a single word: cost. Hardware routers are costly pieces of equipment and represent a large capital investment by operators. Because of this, hardware routers typically have little in the way of spare resources. A clear example of this is the route processor's memory. Modern routers generally have between 256 and 4096 MB of memory [6, 7]. This amount of memory is known to be sufficient for normal conditions, and providing larger amounts of memory undermines cost savings.

In contrast to highly specialized hardware routers, software routers are built using commodity hardware, and have access to the same level of resources any desktop computer does. Software routers are generally a routing daemon running on top of a general purpose operating system. Software routers have the distinct advantage of being cost efficient, costing several orders of magnitude less than hardware routers. However, they lack the high performance line cards and switching fabric of hardware routers, preventing them from handling packet volumes typically found on today's data plane. Because of these qualities, software routers are typically deployed in specialized roles with limited packet forwarding duties.

Traffic traversing routers can be bifurcated into one of two groups the *control plane* and the *data plane*. Network traffic exchanged between routers for the purposes of building best paths to forward data belongs to the control plane. Normal unprivilideged user traffic belongs to the data plane.

2.3 Inter-AS Routing and BGP

The Border Gateway Protocol, or simply *BGP* [8] is the *de facto* standard routing protocol used by inter-AS routers. Throughout this dissertation we will refer to any router running BGP as a *BGP speaker*. BGP allows the exchange of information between ASes about routes to blocks of IP addresses, allowing each AS to have knowledge of options for how to forward packets toward their destinations. Neighboring BGP speakers connect to each other and establish a *BGP session* for the purposes of exchanging path reachability information. BGP is a path-vector routing protocol with policies. This means that

routes contain the path they traverse along with other qualities, and individual routers can define their own policies for which routes are considered "best" and used to forward packets.

These policies frequently extend beyond simply choosing the "fastest" or "shortest" routes: they allow for complex and flexible decisions based on the business relationships between ASes. Since customers pay providers for both incoming and outgoing packets, providers will preferentially forward data toward its destination through customers if able. If the option to generate revenue by handing off traffic to a customer is not available, the AS will attempt to forward traffic through peers since there is no financial cost. Finally, if there is no other option ASes will forward traffic through providers, but will have to pay the provider as a result. In all of these scenarios finding a shorter path is only considered after business preferences are considered.

The act of informing a neighboring router about the existence of a path is called an *advertisement*. A router can advertise a path to any other router it currently has a BGP session with. To do this, it sends a *BGP update* message containing the block of IP addresses reached by the path and a collection of path attributes. Update messages are required to have certain attributes: the path (by Autonomous System number) to the destination, whether the route was learned from a peer inside or outside of this AS, and the next hop in the path. Updates can also contain optional attributes, such as Community Attributes [9].

Upon receiving a BGP update, a BGP speaker will first check to ensure that the update conforms to the BGP specification. After that, a router will test if the route is a loop. This is done by scanning the AS level path for the router's own ASN. If it sees its own ASN, the BGP speaker knows that using this path would result in a forwarding loop. In order to avoid the formation of a loop, the router considers the path invalid and drops the route silently. It should be noted that when doing loop detection a router will only check for it's own ASN, and not check to make sure a loop has already been made. BGP speakers implicitly trust other BGP speakers in the topology to check for the existence of their own ASNs in the path.

The receiving router stores the properties of valid routes inside a Routing Information Base, or RIB. In BGP, routers store all currently valid routes they know about to every destination. Destinations are identified by the block of IP address the path is valid for,

The BGP Decision Making Process	
$r1$	Select the path with best local preference value
$r2$	Select the path with the shortest AS level path
$r3$	Select the path with the lowest Origin attribute
$r4$	Select the path with the lowest Multi-Exit Discriminator
$r5$	Select the path which was learned via a BGP speaker in a different AS
$r6$	Learned via neighbor with smallest ASN
$r7$	Learned via neighbor with the smallest IP address

Table 2.1: The rules BGP uses to select the best path to a given destination.

not the terminal AS. BGP speakers will select for each known destination a single route out of all valid routes for that destination to be considered the best route. That is the sole route that BGP speaker will utilize to forward packets to that particular block of IP addresses. That is also the only route to that destination the BGP speaker will inform its neighbors of.

In order reflect preferences based on business relationships, when a router learns about a path from a neighbor it appends an internal property to that route according to a set of rules configured by the AS called an *import specification*. The property that is used to reflect the business relationship between the AS the router resides in and the AS of the router which advertised the path is called *local preference*. Routes learned from customers will be given the most preferable values of local preference, and routes learned from providers will be given the least preferable, with routes learned from peers sitting between these two groups.

The BGP decision making process, shown in Table 2.1, is a collection of seven rules which are applied iteratively. The goal is to remove routes from consideration as the best until only a single valid route for a given destination remains as a possibility for the best route. BGP speakers will first attempt to find the route with the most preferable local preference value. If multiple routes have the same local preference value, of those routes, the router will select the route with the shortest AS level path length. At that point BGP speakers will only consider routes with the lowest Origin attribute, followed by the lowest Multi-Exit Discrimintor (MED) value. After that the router will attempt to use a path which was learned about from a BGP speaker in a different AS rather than learned from a BGP speaker inside its own AS, this is so called “hot potato routing”. If after all of the previous rules have been applied multiple routes still are in the running to be

the best route, the tie is broken by taking the route learned about via the numerically smallest AS, and if ties still exist, taking the route learned via the router with the numerically lowest IP address. Rules $r1$ and $r2$ address inter-AS concerns. Rules $r3$ through $r6$ address concerns that arise inside the AS the BGP speaker resides in.

BGP speakers are not required to advertise routes to their neighbors, they are allowed to arbitrarily select which of their neighbors they advertise a path to. The rules installed inside a BGP speaker which are utilized to determine if the router will advertise a given path to a particular neighbor are called an *export specification*. Because of the economic implications of AS relationships, an AS will not advertise routes to its providers or peers other than those it or its customers originate. A provider will advertise all routes, regardless of end destination, to any of its customers. These basic policies constitute what is known as "valley-free routing" [10]—an AS never redistributes routes from one of its providers to another; if they violated this, they would end up paying for the privilege of carrying traffic for their providers.

2.4 Path Integrity, Origin Integrity, and BGPSec

BGP does not have any native protection on the integrity of the information stored in AS level paths. Additionally, BGP provides no mechanism for ensuring that a BGP speaker has the right to claim ownership as the end destination of a particular block of IP addresses. This has resulted in numerous incidents where BGP speakers have either accidently or intentionally disseminated false AS level path information or originated routes for destinations they did not own [11, 12, 2]. The consequences of these incidents vary, but generally they resulted in either the inability to forward traffic to impacted destinations or the interception of large volumes of network traffic by ASes which the traffic would not normally be exposed to.

In an effort to prevent such incidents in the future, systems have been proposed which leverage public key cryptography in order to provide guarantees of path integrity and route origination permission. BGPSec [13] is the most mature example of such a system, and the only system which has noticeable efforts made towards deployment. In BGPSec all ASes posses a public key/private key pair. ASes are required to know the valid public keys of all other ASes in the topology. In BGPSec, when a BGP speaker propagates its

best route for a given destination to a neighbor, it sends the BGP update it would send normally, along with a signature on that BGP update, proving asserting that its ASN is correctly append to the path. It also sends along the BGP update which it learned about its best path from, along with the signature of the AS which advertised that path, proving that it legitimately received the route. The BGP update from the prior peer contains the BGP update and signature from the peer before that, this chain goes all the way back to the originating AS. This means that every update contains both the BGP updates and signatures of every AS along the path. By verifying the authenticity of these signatures the receiving router can validate the integrity of the AS level path information. An AS which does not appear on the path can not be added since the signature on an update message adding the ASN to the path can not be generated. Nor can an AS be removed from the path since a gap would appear in the chain of updates.

In order to ensure that only the AS which actually owns a block of IP addresses can originate a path to that destination, BGPSec the numbering authorities, specifically IANA, issue to ASes which control a block of IP addresses a cryptographically signed *route origination authorization*, or ROA. This token binds the ability to originate a path to a particular block of IP addresses to the public key of that AS. ROAs allow the owning AS to originate the listed block of IP address or any smaller block contained inside that block of IP address. They also allow the ROA holder to sub-delegate blocks of IP addresses to other ASes by signing a new ROA for the subblock and binding the new ROA to the receiving AS's public key. The collection of public keys owned by each AS on the Internet, along with currently valid ROAs is referred to as the Resource Public Key Infrastructure, or RPKI. Participation in the RPKI is currently voluntary. Because participation in the RPKI is currently limited, BGPSec is not a currently functional solution. Several of the attacks discussed in this dissertation would either be prevented or need alteration if BGPSec were ever successfully deployed. Where appropriate we will discuss those scenarios, examining how the capabilities of our adversaries would be impacted by a successful roll-out of BGPSec.

Chapter 3

Simulating BGP on an Internet Scale

Evaluating Internet scale attacks is a challenging endeavor. For ethical reasons we can not evaluate attacks directly against the Internet's routing infrastructure. While testbeds such as the DARPA National Cyber Range [14] and DETER [15] exist, access is limited, and researchers are restricted in their behavior. Our solution to this problem was to utilize a combination of simulation and emulation in order to model the behaviors we are interested in.

In this chapter we will briefly cover the challenges in building such simulators. We will focus primarily on challenges to accuracy and scalability. In addition we will cover the two simulators used extensively throughout this work. The first simulator is the Stormcaller [4], which was designed to model the system state individual routers that are part of large BGP deployments. The second simulator we will examine is the Nightwing [5] simulator, which seeks to efficiently model the collective routing decisions of large BGP topologies.

3.1 Challenges in Internet Scale Simulation

The simulators used in this work are essentially large collections of software routers running on a single machine. These software routers, in order to provide meaningful results, need to behave identically to routers deployed on the Internet. They must accurately make routing decisions, which requires maintaining BGP state, being configured with the correct policies, exchanging the appropriate BGP messages, and correctly processing those messages. At a higher level, our simulator will need to arrange the simulated topology accurately, with ASes connected to their neighbors, correct relationships annotated onto those edges, and correct ownership of end networks assigned.

3.1.1 Modeling BGP Topologies

As a starting point, we must have an accurate view of what the network topology is to arrange our simulated BGP routers. The first observation we can make is, assuming we are only concerned with inter-AS behavior, we can largely ignore the internal topology of ASes. BGP, requires all BGP speakers residing in the same AS to have essentially identical state. This is either accomplished via having all BGP speakers inside an AS connected to each other in a mesh, or through the use of a special type of software router

called a router reflector, which broadcasts BGP messages to every BGP speaker in an AS. Because of this configuration, each router inside an AS will have to process the same number of BGP messages, and in the overwhelming number of cases will come to the same routing decision. This allows us to represent each AS via a single representative router.

To build our Internet topology we consult the CAIDA inferred AS relationship dataset [16]. This dataset, built off of observations of routing tables from the Internet, provides us with two vital pieces of information. First it provides us with a list of AS adjacencies. This allows us to connect our simulated routers together to the correct set of BGP neighbors. Second, this dataset provides us an indication of the inferred relationship between the two adjacent ASes. This information will be used to correctly configure router policies, discussed in Section 3.1.3.

3.1.2 IP Block Ownership

Another challenge is assigning ownership of the end networks, specifically the IP blocks which BGP operates on, to their appropriate ASes. While IANA, and by extension the region numbering organizations, are responsible for delegating blocks of IP address to specific ASes, they do record an further information on sub-delegation or resale of IP blocks to different ASes. As mentioned in Section 2.4, the RPKI attempts to provide an accurate view of who is allowed to originate specific blocks IP addresses, however participation is optional and at the time of this work the accuracy of information contained in the RPKI was questionable.

In order to build our view of what AS owns each block of IP addresses, we consulted the RouteViews dataset [17]. RouteViews is a collection of BGP speakers which network operators voluntarily peer with, which provides researchers with valuable insights into the current state of routing tables on the Internet. In order to build IP to owning AS mappings, we analyzed the state of RouteViews routing tables for a month before and after our inferred AS relationship dataset was generated. We examined what AS each BGP speaker peering with a RouteViews router believed originated a block of IP addresses. In situations where multiple ASes were observed we selected the most commonly seen originator of the IP block.

3.1.3 Router Policy

As discussed in Section 2.3, BGP's route selection and router advertisement process is primarily driven by the relationships between ASes. We utilize the inferred AS relationships from the CAIDA dataset to provide the AS relationships a router's import and export policies are configured on. Import and export policies are based on commonly held operator best practices. Because the exact configurations of BGP speakers are a closely held secret of ASes, we can not fine tune to scenarios where, for business reasons, an AS would prefer one neighbor of a particular relationship over another neighbor of a similar relationship. In our simulator, all customer learned paths are assigned an identical, highly preferable, local preference. Similarly, all peer and provider learned routes are assigned local preferences that identical to other paths learned from different neighbors with the same relationship status. Export policies are configured to always advertise all customer learned routes, and to advertise all learned routes to customers.

A general concern is the accuracy of our simulated paths. Because of the actions taken by some adversaries covered in this thesis, specifically those of Chapter 6 and Chapter 7, we can not take an approach similar to other recent work, for example RAPTOR [18], where we only use routes observed in the wild. This is because later adversaries will use routes which exist, but are not currently utilized because they are suboptimal. Because they are not currently selected as the best route they will not be advertised, which mean that we can not observe them directly. We parsed information from RIPE route reflectors [19] and RouteViews peering points [17] within a three month window before and after when our inferred AS relationship datasets were generated by CAIDA. For the routes our simulator considers best, we should generally find a majority of those routes appearing in the wild at some point in time. This in fact is true, we find that the majority of our simulated paths are seen in the wild, specifically our simulator aligns with Anwar *et al.*'s [20] results that approximately two thirds of routes should be correctly predicted. Interestingly, many of the suboptimal routes our simulator predicts as existing for use by our later adversaries can also be observed in the wild at some point in time. A full table of the fraction of simulated routes for ASes residing in nation states which are particularly interesting to this work can be seen in Table 3.1.

	Simulated Best Observed	Suboptimal But Existent Observed
Brazil	63.2%	46.6%
China	62.2%	42.2%
Germany	56.4%	49.1%
Iran	45.3%	13.4%

Table 3.1: Measurement of the accuracy of our topology compared to actual observed routes collected a route reflectors around the Internet. The second column is lower as a result of many of the existent, but suboptimal routes violating no Valley Routing.

3.1.4 State and Computation Management

One of the primary problems with simulating Internet scale phenomena is executing the computations for routers from every single AS on the Internet, along with maintaining the state to do those computations correctly. While BGP messages can not be executing in parallel in the context of a single router, in our simulations we can execute the computations of different BGP speakers in parallel as they are independent of each other. Both of our simulators rely on this parallelism in order to run with reasonable efficiency.

As mentioned in Section 2.2, routers have between 256 MB and 4096 MB of general purpose memory, the majority of which is dedicated to storing the state needed to execute the BGP decision making process. Given that there are roughly $65,000$ ASes in the Internet, if we directly emulated full routing tables for all ASes we would quickly experience a memory requirement out of scope for modern hardware. An estimate of memory demands for a simulator that directly emulated how BGP stores routes can be seen in Figure 3.1. Therefore, we must find a way to shrink state while not losing computational accuracy. Two optimizations will allow us to run simulations with between 64 and 240 GB of RAM. First, we can observe that the majority of ASes are only connected to a single other AS, which is their provider, these AS are termed a *single homed* ASes. Since single homed ASes have only a single choice for all paths, we can essentially prune them from the topology, and absorb them into their single provider AS. Whatever decision the provider makes in terms of the best path will be the single homed AS's only choice. This pruning means that we will only need to simulate between $7,000$ and $18,000$ ASes rather than $65,000$. Second, while BGP operates on a per block basis, we can see that

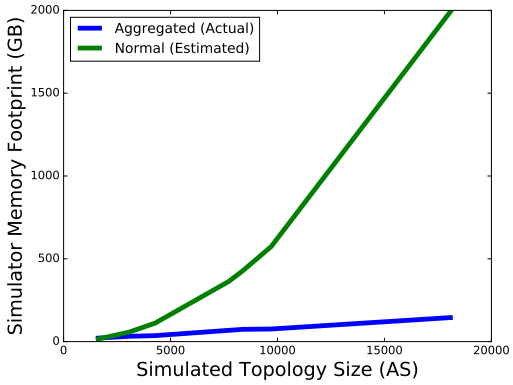


Figure 3.1: Comparison of the memory footprint of the Nightwing simulator when aggregated vs non-aggregated routing tables were used for various sizes of network topology.

given how our simulated BGP speakers are configured, the same path selection decision will be made for all IP blocks originated by the same AS. This allows us to collapse the IP state down from several hundred thousand IP blocks down to several thousand, a two order of magnitude improvement. The result is a much more manageable simulator memory footprint, shown in Figure 3.1.

3.2 The Stormcaller Simulator

The goal of the Stormcaller simulator [4] is to take the general design choices made in Section 3.1 and build a simulator capable of accurately modeling the dynamics of a system of BGP speakers. Specifically we are interested in low level details of each router, for example the current CPU utilization, pending BGP message queues, and amount of free memory to name a few. More specifically we are interested in knowing how these details change over time. Stormcaller is an event driven simulator that seeks to accomplish this. Stormcaller is configured to compute processing speed and memory consumption based on observations of real world hardware and software routers we had direct control over. Figure 3.2 shows a comparison between a real timed injection of a global routing table into a CISCO 7603 router and the same router as simulated by Stormcaller.

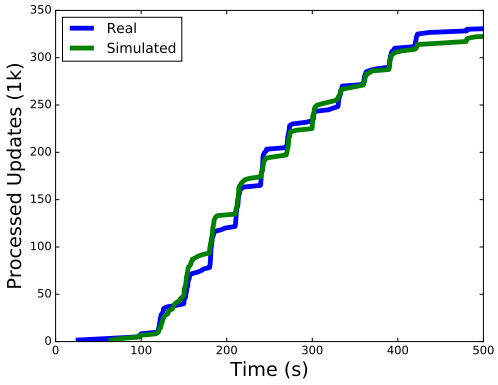


Figure 3.2: Comparison of the time required for a real CISCO 7603 router to handle the injection of a sample global routing table versus a simulated router in the Stormcaller simulator.

Because the processing speed and memory consumption of routers is strongly influenced by the actions of adjacent routers, care was needed to ensure that the simulator correctly handled behavioral dependencies. In order to achieve a high degree of parallelism Stormcaller analyzed the topology and current state of routers, looking for key changes in behavior which would impact the processing state of routers, for example a BGP speaker starting or stopping the advertisement of paths to neighbors. Stormcaller would then compute the area of the topology that would be impacted by this event, something termed the dependency region. It should be noted that dependency regions can overlap. Dependency regions are then sorted according to when their event occurs. The region with the nearest event is executed in parallel up to the event. Speculative execution was also employed when CPU utilization was low to execute multiple non-overlapping dependency regions, and in scenarios where a later dependency between the two regions is detected roll-back of the later region is executed. In order to minimize this heuristics involving the delta between dependency region event time and separation of the two dependency regions were employed.

3.3 The Nightwing Simulator

In contrast to the Stormcaller Simulator, the Nightwing Simulator [5] is concerned only with the routing decisions a BGP topology will make after full network convergence. This means that we are no longer required to concern ourselves with accurately modeling *when* decisions are made, only what the decision is. Nightwing was also intended to be extensible, allowing for modules to be added easily which captured how other outside agents, for example censorship circumvention systems, the actual hosts contained inside ASes, and the business portion of the ASes themselves, behaved in reaction to the converged BGP topology.

Rather than being event driven, this simulator executes in a round based manner, allowing for maximum computational throughput. In each round, each router processes all of its pending BGP messages, and only sends BGP messages to peers to update state at the end of a computational round. This means that if a route to a given destination changes multiple times inside a single round of computation, only the final state is forwarded to peers, this saves us un-necessary computation since we are only concerned with routing decisions at convergence. Nightwing simultaneous executes the BGP decision making process for multiple ASes, allowing it to scale with the number of available cores. However, decisions for a single AS are done in serial, this allows us to rely on existing proofs of correctness about the BGP decision making process.

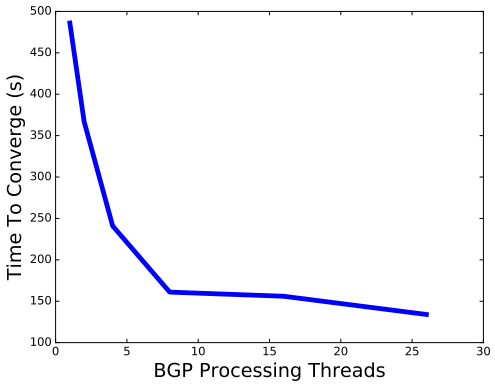


Figure 3.3: The time required to build a full simulated Internet topology as a function of the number of threads available to Nightwing. The machine in question had access to 32 total cores.

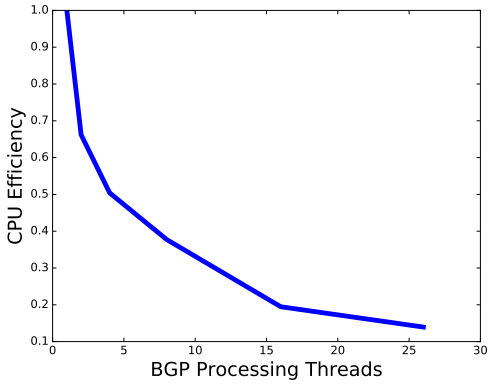


Figure 3.4: The CPU efficiency of Nightwing. The deteriorating efficiency is a result of asymmetric work loads between ASes.

On hardware used for the later simulations in this work, the routing decisions of every AS in the Internet could be computed on the order of two to three minutes depending on the number of cores available. Figure 3.3 shows the time the Nightwing simulator takes to converge as a function of available cores. The machine this graph was generated on had 2 Intel Xeon E5-2630 processors which each have 8 cores, giving the machine direct access to 16 hyper-threaded cores. One thing that jumps out from Figure 3.3 is that doubling the number of threads executing Nightwing does not always result in a halving of the runtime. We can formalize how well Nightwing uses cores by considering the metric of CPU efficiency. CPU efficiency is defined as follows:

$$CPUEfficency = \frac{SpeedUp}{NumberofThreadsUsed} \tag{3.1}$$

$$SpeedUp = \frac{SerialRuntime}{ParallelRuntime} \tag{3.2}$$

For Nightwing this is plotted in Figure 3.4. We can see that after roughly 5 threads the gain begins to fall off, with any additional threads past 16, the number of cores the program has access to, providing very little additional benefit. The reason for this falloff in performance benefit is the asymmetric nature of computational burden between ASes. Certain ASes have far more processing requirements than others, meaning that their computations for a given round will take longer. Recall that processing of the

updates for a single AS happens on a single core. In general what we find is that each round a few ASes take markedly longer than others, causing cores other the ones computing those ASes to idle. In the future, if more efficiency is needed, investigation could be undertaken looking at decomposing the computations inside a single AS across multiple CPUs.

Chapter 4

Adversarially Induced BGP Instability and the Cross Plane Session Termination Attack

In this chapter we will explore how an adversary who does not directly control any BGP speakers can still manage to disrupt the functionality of an entire BGP topology. Specifically, we will discuss the Coordinated Cross Plane Session Termination, or CXPST, attack, a form of distributed denial of service (DDoS) attack that attempts to exploit the global scope of BGP updates to induce control plane instability on the Internet as a whole. In order to artificially create control plane instability, CXPST applies Zhang et al.'s [21] work on disrupting BGP sessions between routers. This prior work described how an unprivileged adversary with only the ability to send ordinary data traffic can exploit the fact that the control plane and data plane use the same physical medium. This co-location allows an adversary to convince a BGP speaker that one of its BGP sessions has failed. CXPST computes centrality measures of the network topology and uses this information to intelligently select a collection of BGP sessions to disrupt using Zhang et al.'s attack. This results in waves of control plane instability which, because of the choice of links, are broadcast globally. By exerting influence over the location and times of failures, CXPST generates enough updates to overwhelm the computational capacity of routers, crippling the Internet's control plane.

In Section 4.2 we examine a collection of technical challenges an adversary would need to solve in order to launch such an attack, presenting solutions for each challenge. In Section 4.3 we will evaluate the effectiveness of our solutions. There are three main challenges we will focus on in these two sections. We will need to demonstrate that an adversary has the ability to successfully attack specific BGP sessions at arbitrary points in the topology. We will then need to ensure that target points in the topology are selected to maximize the control plane instability seen in core routers resulting from BGP's natural reaction to these failed sessions. Lastly, we examine if these failures can be coordinated such that the aggregated impact of this instability prevents routers from making path selection decisions in a timely manner.

We also consider defenses against CXPST in Section 4.4. We demonstrate that currently-deployed mechanisms to combat control plane instability would be ineffective against CXPST. Instead of attempting to stop BGP from broadcasting updates globally, we focus our defenses on preventing an adversary from disrupting BGP sessions, presenting a short term, easily implementable solution functions with partial deployment. We also discuss long term defenses, which require redesigning routers.

What is most troubling about the attack outlined in this chapter is that, unlike Coremelt [22] and other Internet-scale DDoS attacks, CXPST does not directly attack *all* links on the Internet. Instead, CXPST will only attack a small subset of links, using the properties of these links to amplify the attack. This reduces the bandwidth required to successfully launch CXPST compared to related attacks. Through simulations we will demonstrate that botnets on the order of $250,000$ members can cause severe disruption to the Internet control plane, even under conservative estimates of adversarial bandwidth and router over-provisioning. The resources to launch this attack are now widely available, due to the explosion in the number of end-user machines that have been compromised by a centrally controlled virus or worm. These botnets provide access to massive amounts of distributed computing power and aggregate bandwidth of several terabits per second [23, 24]. This lower than expected resource requirement is especially troubling given simulations which show that core routers will experience processing delays on the order of minutes rather then microseconds after a few minutes of an adversary launching CXPST.

4.1 Background

4.1.1 BGP Stability and Network Performance

In normal BGP operation the network converges to a stable state. However, local changes such as cable cuts, router hardware failure, or changes in local BGP policy can result in routes having to be withdrawn, leading to routing table recalculation and re-advertisements to other routers. The catch is, these advertisements can lead to the same activities on the routers that receive them. This includes advertisements to yet more routers who will repeat this process, possibly causing the update to be propagated globally. This behavior demonstrates a key fact: in BGP small local changes are often seen globally.

Instability in the control plane can reduce the performance of the data plane [25, 26, 27]. When a router is shut down, paths that pass through that router will no longer function, and new routes must be found. Functioning routers will continue forwarding traffic towards the now non-existent router until they complete the process of finding a new route. All traffic directed toward the powered down router will be lost, resulting in

large volumes of dropped packets. This is just one example of how instability can result in large disruptions of the data plane.

When a set of routes oscillates rapidly between being available and unavailable it is termed *route flapping*. Route flapping can be the result of several flaws in the network, including misconfiguration, faulty router hardware, and link failures. It is a problem because of the sheer number of control plane messages generated, and the resulting routing table re-computations that routers must perform. Data plane performance is only restored after affected routers complete the processing of BGP messages. In the case of large amounts of instability, route re-computation increases the load on a router's CPU dramatically, potentially exceeding its capacity. This increased load translates into a longer turnaround time for processing decisions, which in turn extends the duration of the data plane disruption. During route flapping, routes need to be recalculated as quickly as possible, but the fact that so many routes need to be recalculated slows that computation.

Some functionality exists currently that attempts to mitigate the damage route flapping does to both the data and control planes. Minimum Route Advertisement Intervals (MRAI) [8] prevent a series of rapid advertisements of route changes for the same network. While MRAIs do not help the data plane recover directly, they do reduce the load on a router's CPU. BGP Graceful Restart [28] provides a grace period where two connected routers allow their data planes to continue functioning even while there is an issue on their control plane. This attempts to mitigate issues where two routers need to recover from a simple error. Lastly there is Route Flap Damping [29], which directly aims to combat route flapping by suppressing (ignoring) routes that exhibit flapping behavior. The initial work to route around failures still needs to be done, but additional work is not done as the link oscillates between functional and non-functional states.

4.1.2 Attacks on BGP Routers

Given the importance of routers and routing protocols, it is unsurprising that there exists a large body of literature exploring their weaknesses. Of particular interest to this work is a paper by Zhang, Mao, and Wang [21] that looks at using brief targeted data plane congestion to trick a pair of routers into disconnecting from each other. In their attack, which we will refer to as the *ZMW attack* for the rest of this Chapter, an unprivileged

adversary indirectly interacts with the control plane via the data plane. This is possible because the data plane and the control plane are co-located. Because of this co-location, congestion from data plane traffic can cause the loss of control plane traffic. There are several places inside a router where control plane traffic and data plane traffic contend for resources, including buffer space and bandwidth. When resources are scarce, control traffic and data traffic must share these limited resources.

The BGP protocol (see Section 2.3) uses hold timers as one way to detect a failed session. Routers keep track of the last time they received control plane data from a BGP peer, and, if this time exceeds the hold timer, the session is torn down. If enough consecutive control plane packets are lost, the hold timer of a BGP session will expire and the session will fail. In essence an adversary can use a flood of data to digitally "cut the link" between two routers. When the BGP session fails, all routes discovered via that session will have to be withdrawn and new routes recalculated on both sides of the "failed" link. Zhang et al. demonstrated in both hardware and software routers the ability to successfully implement this attack.

4.1.3 Botnets

A common trend in malware is the distribution of bots that are under the control of a central entity. The infection propagates using known techniques including trojan horses and vulnerability exploits. After a successful propagation, the attacker can be left in control of a set of hosts numbering upwards of several hundred thousand nodes [24]. The propagation techniques may affect the distribution of the bots and we discuss the issue in our attacker model. Having control over a large botnet gives an attacker control over multiple distinct traffic sources and sinks which can be used for attack traffic generation or as vantage points to map Internet topology.

Network disruptions during large malware infection events have been documented in the past, including the Code Red and Code Red 2 instances [30] , the Nimda worm [30] , Slammer [31] and Blaster. In Code Red's case, the resulting large set of malware instances targeted a single site (www.whitehouse.gov), but was otherwise uncoordinated. In at least the Code Red, Nimda and Slammer events, instabilities in the control plane including cascading failures during the infection events have been reported [32, 30].

4.2 The CXPST Attack

In this section we present CXPST, an attack against the Internet's control plane. In CXPST, an adversary in control of a botnet selectively disrupts BGP sessions in an effort to artificially generate a large number of BGP updates. This surge of updates overwhelms the computational capacity of routers, preventing them from efficiently making routing decisions.

4.2.1 Attacker Model

There have been many instances of adversaries causing control plane instability by intentionally misconfiguring routers under their control [33, 34, 11, 35]. These attackers were able to to interact with the control plane directly using their privileged status as BGP speakers. Attacks at this level can be typically prevented with the use of BGPSEC or similar technologies [36, 37, 38].

In this work we instead consider an unprivileged adversary who does not control any BGP speakers, and consequently can only create data plane traffic. Lacking the ability to directly generate control plane messages, these adversaries instead need to force non-colluding routers to generate control plane events. We specifically consider an adversary who controls a botnet of reasonable size. This attacker is capable of generating network traffic from compromised hosts distributed across the Internet. Adversaries in control of compromised BGP speakers would be capable of generating some of the phenomena used to drive CXPST, but would be unable to do so at arbitrary locations in the network.

4.2.2 CXPST Conceptually

In order to create control plane instability, our attacker will apply the ZMW attack [21]. As discussed in Section 4.1.2, ZMW uses data traffic to trick a pair of routers into disconnecting from each other. This results in a set of route withdrawals, recalculations, and advertisements. Interestingly, the control plane disruption generated is not limited to the one set of withdrawals and advertisements. Since the targeted link is no longer used by routes after the BGP session fails, no traffic will utilize the link. This allows the two attacked routers to communicate with each other once more, as the link will no longer be congested with attack traffic. The targeted routers will, after a small amount

of time, re-establish their BGP session. This will result in further BGP updates as the routes that were just withdrawn are re-advertised. Bot traffic will once again shift to the targeted link as the previous routes become utilized once more, and the attack resumes without any intervention from the attacker. The targeted BGP session will again be destroyed and the cycle repeats itself, forcing the targeted links to oscillate between "up" and "down" states. In essence, CXPST induces targeted route flapping.

While the two routers attacked will be most impacted, routers not directly attacked will be affected as well. As mentioned in Section 4.1.1, BGP updates that result from local changes tend to be broadcast on a global scale. By creating a series of localized failures that have near global impact, CXPST has the potential to overwhelm the computational capacity of a large set of routers on the Internet.

There are three key tasks that CXPST needs to accomplish in order to function. First, the correct BGP sessions must be selected for attack. These BGP sessions must be selected to maximize control plane instability when they fail. If an insufficient number of BGP updates are generated, then routers will not be computationally exhausted, and the attack will not succeed. Second, the attacker needs to direct the traffic of his botnet onto the targeted links. While Zhang et al touch on this in their work, they do not deal with the difficulties in managing attack traffic on a dynamic network. For example, congestion on links used to approach targeted links must be minimized. Link failures on the way to the target will prevent attack traffic from reaching its destination, possibly preventing the termination of the targeted BGP session. Lastly, since CXPST is essentially route flapping, the attacker must find a way to minimize the impact of existing mechanisms that attempt to mitigate the effects of route flapping.

4.2.3 Selecting Targets

Maximizing control plane disruption is equivalent to maximizing the number of BGP update messages that are generated as a result of link failures. Centrality measures from graph theory provide a good starting point for building a heuristic to govern target selection. Our method of selection uses a slightly modified version of edge betweenness as a metric. Normally edge betweenness is defined as:

$$C_B(e) = \sum_{s \neq t \in V} \frac{\sigma_{st}(e)}{\sigma_{st}} \tag{4.1}$$

where σ_{st} is the number of shortest paths between nodes s and t, and $\sigma_{st}(e)$ is the number of those paths that contain the edge e. BGP does not always use the shortest path between two ASes however. Because of this we use a modified definition of edge betweenness:

$$C_B(e) = \sum_{s \neq t \in V} path_{st}(e) \tag{4.2}$$

where $path_{st}(e)$ is the number of BGP paths between IP blocks in s and t that use link e. Since each of these routes must be individually withdrawn, recomputed, and re-advertised this will provide an approximation of the number of BGP messages generated if the link were to fail. Consequently, target links are ranked in order of their "BGP Betweenness".

Another reason to use BGP betweenness is that our attacker possesses the resources to measure it. As stated in Section 4.2.1 our attacker controls a botnet distributed across the Internet, this provides him with a large number of distinct vantage points. Prior to the attack, bots can perform traceroutes from themselves to a large set of nodes in separate networks and report the results. By aggregating the results an attacker can generate a rough measure of the BGP betweenness of links. Each time we see an edge in our aggregated traceroute data set, it represents an individual route that crosses a given link. This is because each traceroute originates from a distinct source and travels to a distinct destination.

Equal cost multi-path routing, or ECMP, presents an additional issue for CXPST. It requires markedly more resources to congest a link when it is part of a set of load balanced links then when it is a stand alone link. In order to avoid this, when traces are being gathered, multiple traces need to be taken. These traces can be compared in an effort to detect ECMP. Any links that are possibly using it are removed from the set of potential targets. Recent studies [39] have shown that load balancing, while prevalent inside ASes, is not widely used between ASes. This is a best case scenario for our attacker. Load balancing inside ASes removes potential bottlenecks for attack traffic, and since CXPST only attacks links between ASes, few targets will be excluded.

4.2.4 Attack Traffic Management

At first glance, selecting which bots will attack a given link appears straightforward. As discussed by Zhang et al. [21], an attacker could simply use all bots that can find some destination such that the path to the destination crosses the targeted link. This method suffers from two main weaknesses. First, this strategy fails to take into account the fact that network topology is dynamic. This issue is especially important in the case of CXPST as the attack forcibly changes network topology in multiple places. Second, there is the possibility that we will saturate bandwidth capacity on the way to the target link. This can result in the unintentional termination of BGP sessions, cutting off our path to the target.

Dealing With Changing Topology

CXPST actively changes network topology. The attacker must select which bots will attempt to attack a given link with this in mind. Instead of simply checking that a given path contains the target link, the attacker must ensure that the path does not contain other links that are being targeted as well. By doing this, when links targeted by CXPST fail, attack traffic will not be re-routed.

Attack traffic can still be re-routed because of the unintended disruption of a non-targeted link. In order to counter this, an attacker should send more attack traffic toward a targeted link then is needed to congest it. This "safety net" will allow some amount of attack traffic to be diverted because of network dynamics without relaxing pressure on targeted links.

Fixing the Flow Issue

Our attacker will typically have more bots able to attack a given link than needed. Care must be taken when selecting a subset of these bots to attack the link. In order to minimize the amount of congestion prior to reaching the targeted link, the attacker should keep the attack traffic dispersed until it reaches the target. When the attack traffic reaches the targeted link the attack flows will be aggregated together, causing congestion on that link. After the intersection point traffic takes different paths toward its final destinations, dispersing in an effort to not congest downstream links. This

is shown in Figure 4.1, where the link between t_1 and t_2 is the targeted link. Traffic approaches from a variety of sources, heading to a variety of destinations. Traffic levels on links before and after the target are not substantial. For example the link between s_1 and t_1 or the link between t_2 and d_2, are manageable, but the aggregation of flows across the t_1 to t_2 link creates congestion.

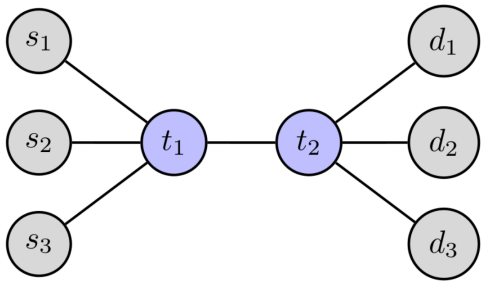


Figure 4.1: An illustration of attack traffic aggregating. $s_1 \ldots s_3$ are source ASes, $d_1 \ldots d_3$ are destination ASes, and t_1 and t_2 are targeted routers.

CXPST uses a straight-forward algorithm to automate attacker assignment. Prior to allocating resources, our attacker builds two flow networks based on the traceroutes used to select targets. In one network, bots are treated as sources and target links are treated as sinks. In the other, target links are treated as sources and destination networks are treated as sinks. The attacker can either guess the bandwidth of links involved or actively measure their capacity. When selecting destinations for attack traffic, the attacker runs a max flow algorithm on the first flow network, establishing which bots will be used to attack each targeted link. Then the second flow network is then analyzed to determine which destination networks attackers should address their traffic to. Where possible bots will attempt to send attack traffic to IP address of other bots in the botnet as described by Sunder and Perrig in Coremelt [22]. In this way, traffic sent by the attacker is "wanted" and not flagged as malicious by end hosts.

4.2.5 Thwarting Defenses

As was mentioned in Section 4.1.1 there are some mechanisms that exist to reduce the effects of route flapping. Since CXPST is artificially induced route flapping, these defenses might impede it. These defenses though, were designed to deal with random network events, not an adaptive adversary. Two of the defenses, BGP Graceful Restart and Minimum Route Advertisement Intervals, require no changes. Route Damping on the other hand requires some minimal changes to CXPST's behavior. During the course of the attack the bots will need to remove links that get damped from their target set. Bots notice that links are being damped when the paths used to reach their targets do not re-appear within a time window. New target links are then chosen from the list of available targets. We will demonstrate CXPST's ability to function in the presence of these defenses in Section 4.4.

4.3 CXPST Simulations

There are a large number of questions to be asked of CXPST. Will real world bots be in a position to send traffic over a given link? Will bots over-saturate the edges of the network before reaching their target? How many BGP updates would CXPST be able to generate? Would the rate of these updates be sustainable over the duration of the attack? What would the impact of these updates be on routers? In this section we will answer these questions with the assistance of simulations using the Stormcaller simulator.

4.3.1 Simulation Details

In this section we will discuss some of the configuration choices for the simulations along with any simulation details specific to these experiments. The base simulator is the Stormcaller simulator discussed in Chapter 3. Additional, details of the bandwidth model used by links and the distribution of bots in the simulated botnet are also covered in this section.

Network Topology

The topology utilized in these simulations are built using inferred AS relationships from January 2010. The full AS level topology was pruned to contain only ASes that provide service to other provider ASes, i.e. all the ASes who had at least one customer that itself had customers. A graph was then generated containing these ASes and any edge that existed between ASes in the subset. The result was a connected graph with 1829 ASes and nearly $13,000$ edges. The ownership of the IPv4 space is harvested from Routeviews routing tables taken from the same month.

Unlike other simulations using the Stormcaller simulator, the properties of the links connecting ASes are important for these simulations. In this simulation model only a single link connects two ASes. In reality there are three possible scenarios for an inter-AS connection. First, there might indeed only be one link. Second, there might be more then one link, but only one is actively used for traffic, or at least traffic originating for a given area. Third, there are multiple links and they are all actively used. Since CXPST only attacks individual links (compared to other DDoS attacks, for example Coremelt [22], which target *all* links in an AS) we can elect to represent edges in our topology by a single link. This is accurate in the first two scenarios previously mentioned (when a single links serves a geographical area bots can be selected from just that area in order to target it). In the third case, multiple active links, our assumption would be inaccurate, but as stated in Section 4.2.3, our attack actively avoids load balanced links. The fact that we don't need to simulate attacking these links, coupled with the previously mentioned fact that these links are uncommon [39], means that this simplification is acceptable.

The bandwidth model for links in these simulations is meant to be as disadvantageous to the attacker as possible. Link capacities are based on the degrees of the connected ASes. Since we are concerned about the ability to fill core AS links we use OC-768 size links, the largest link size currently in the SONET standard, for those links. In the same spirit we connect all fringe ASes, where the majority of the attacker's resources reside, with OC-3 links. It is important to mention that while the aggregate bandwidth between two ASes may be much higher than a single OC-768 link, we are only concerned with attacking *single inter-AS links*, meaning that having to attack an OC-768 link is truly a worst case scenario for an attacker.

The Botnet

Along with topology, bot placement also impacts simulation results. Papers on botnet enumeration have given some insight into the distribution of bots throughout the Internet, allowing us to use a real bot distribution in our simulator. We used the data set for the Waledac botnet [40] to build our model of bot distributions. IP addresses of infected machines were mapped to their parent ASes using the GeoIP database [41], providing a rough count of infections per AS. We then uniformly scaled these numbers up or down to achieve the botnet size desired. To ensure a proper lower bound for attacker bandwidth, bots were given a basic ADSL connections with an upload capacity capped at 1.0 Mbit/sec [42]. Bots were only given the ability to send network traffic and perform traceroutes. They were *not* given any additional information about the network, such as link capacities or AS relationships.

4.3.2 Simulating CXPST

Our event driven simulator allows us to view the results of a botnet executing CXPST. At the beginning of a simulation, routers are allowed to connect to their BGP peers and reach a stable network state. After the network has reached a stable state, bots are allowed to interact with the network. Bots only have the ability to run traceroutes and to send network traffic. The ability of bots to send traffic is limited by the bandwidth of links carrying the traffic. All routers in the simulation are vulnerable to the ZMW attack, meaning that accidental disruption of BGP sessions in the simulation is possible. This may lead to traffic redirection away from targeted links.

The impact of CXPST needs to be evaluated in three places. First, we must answer the question of how bot placement and bandwidth bottlenecks affect the ability to attack specific links. We can compare the number of targeted and un-targeted BGP sessions that are disrupted during the course of the attack. This will give us a grasp of the feasibility of attacking specific BGP sessions.

Next, we examine the effect of these disrupted BGP sessions. Apart from topology changes generated by CXPST, the topology of our simulator is stable during the course of the simulation. This means that any updates seen are a direct result of the attack. Our simulator logs the arrival of BGP messages to routers, giving us a record of the

number of BGP updates generated by CXPST. We can compare the number of update messages generated to normal loads providing us with a measure of CXPST's success.

Lastly, we would like some idea of the impact any dramatic increases in BGP update rates would have on the routers themselves. One measurable effect is the increase in the time between when a router receives a BGP update message and when it is finally processed. If the time to process an update becomes large, the data plane suffers dramatically, as local outages are not reacted to and traffic is sent to dead links. Using the logs of BGP update message arrivals and a benchmarking study by Wu et al. [43] we can build an estimate of the time to process BGP updates during an attack.

4.3.3 Simulation Results

CXPST was simulated with botnets of 64, 125, 250, and 500 thousand nodes. We describe the results of these simulations in this section. In general the majority of our testing focused on the 250 thousand node botnet scenario. Diminishing returns from increasing botnet size drove this decision.

Success in Disrupting BGP Sessions

We can examine our ability to successfully disrupt only targeted BGP sessions by placing them into buckets. We chose three different descriptors for links: targeted links, last mile links, and transit links. Any link selected for disruption by CXPST is considered a targeted link. Last mile links are un-targeted links that connect fringe ASes to the rest of the network. Any link that does not fit the other two categories is considered a transit link. Our attacker's goal is to maximize the number of targeted links that fail while minimizing the number of failures in the other two categories. As mentioned in Section 4.2.4, CXPST sends more attack traffic to a link then is needed to hedge against traffic loss. In our simulation an extra 30% over the estimated required traffic was sent.

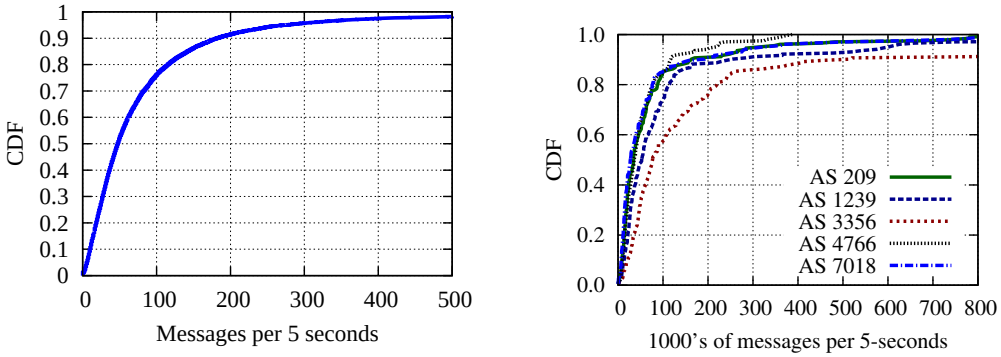
The results of a $250,000$-node attacker can be seen in Figure 4.2. Our simulated attacker successfully disrupts more then 98% of targeted links during the course of the attack. The attacker only disrupts roughly 19% of last mile links at some point in the attack, far less then the 30% that is tolerable with the safety net. Most importantly, less then 4% of transit links are disrupted during the attack. This demonstrates the ability of CXPST to surgically disrupt BGP sessions.

Figure 4.2: Percentage of BGP sessions for various types of links that failed when a botnet of 250 thousand bots launched CXPST. Note that a 30% "safety buffer" was used, so that up to roughly 30% of last mile links could fail without impeding CXPST.

BGP Update Generation

Next we will examine the number of BGP updates that are the direct result of CXPST. We are most concerned with the impact on *core routers* in our topology, the top 10% of ASes by degree. Emphasis should be placed on core routers because of the potential impact on the rest of the network. These ASes are utilized by the majority of other ASes as transit providers, and instability in these routers would be felt across the Internet.

Using simulation logs, we gathered information on the number of BGP updates routers receive during 5 second windows of time. We turned to the RouteViews data set [17] to get an idea of baseline router load. In excess of 23,000 network operators voluntarily start BGP sessions with RouteViews routers in order to validate their network configuration from an outside vantage point. RouteViews routers keep a log of the real time arrival of BGP update messages from these sessions. We used logs from January 2010, the same month as our AS relationship data, to build a view BGP update load. Figure 4.3.3 shows a CDF of the number of messages a RouteViews routers see per 5 seconds window. It is important to note that RouteViews routers have an inordinate number of BGP sessions relative to edge routers in transit ASes, meaning that this number of updates is more then likely an overestimation of the number of messages seen by a BGP speaker, and consequently will result in an *underestimation* of our attack's effectiveness.



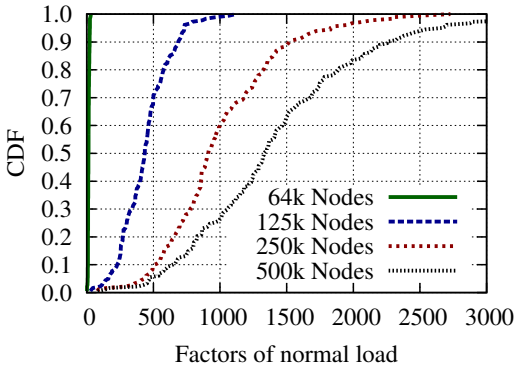
(a) Normal Load

(b) Load Under Attack

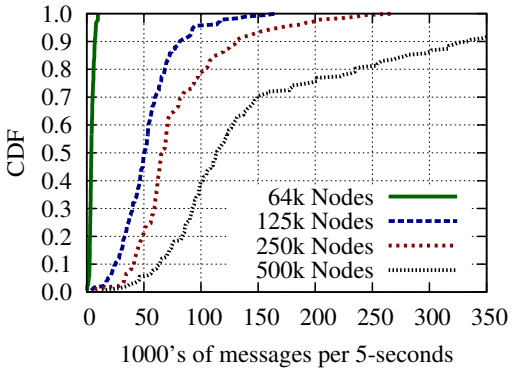
Figure 4.3: A CDF of normal BGP update loads for a RouteViews router during January 2010 on the left, compared to loads for a collection of specific AS under attack by 250,000 bots on the right.

As was mentioned in Section 4.1.1, large bursts of updates have a significant impact on the performance of the Internet. Simulations show that CXPST successfully creates BGP update message bursts throughout the duration of the attack. For example, during normal operation (see Figure 4.3.3), the 90th percentile load is 182 messages per 5 seconds. During CXPST the 90th percentile load is dramatically increased for the targeted routers, a CDF of their 90th percentile loads is shown in Figure 4.4c. In the case of the 250,000-node attacker, more than half of core routers are at or above a four order of magnitude increase in load. These bursts of updates are not a few isolated incidents. At the 75th percentile of update load, shown Figure 4.4b, we continue to see the same dramatic increases in processing load.

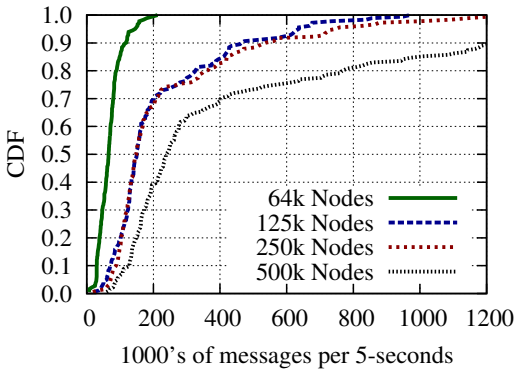
Moreover, these spikes are not the only effect of CXPST, an increase in BGP update rate is felt throughout the attack. Figure 4.4a shows the increase in the median load of routers during the attack. In the case of the 250,000-node botnet, the median load on nearly half of the core routers increased by a factor of 800 or more. Even using the 125,000-node botnet results in 50% of routers' median loads increased by a factor of 400 or more. This increased median load shows that routers will not have a chance to recover from the previous bursts of updates. Figure 4.3 plots distributions of message loads for several large ISPs during the simulated attack by 250,000 bots.



(a) Median Load



(b) 75th Percentile



(c) 90th Percentile

Figure 4.4: Median router load of targeted routers under attack as a factor of normal load 4.4a; and 75th percentile 4.4b and 90th percentile 4.4c of message loads experienced by routers under attack, measured in BGP updates seen in 5-second windows.

Time to Process Updates

The end results of CXPST can be seen by examining the time required to process a BGP update message. Routers process BGP update messages at a roughly constant rate. If the rate they are received at surpasses the rate of computation, messages will need to be buffered, and processing delays will occur. Using performance figures from a router benchmarking study [43] we computed the delay between when core routers received BGP updates and when they finally finished processing those updates while under attack.

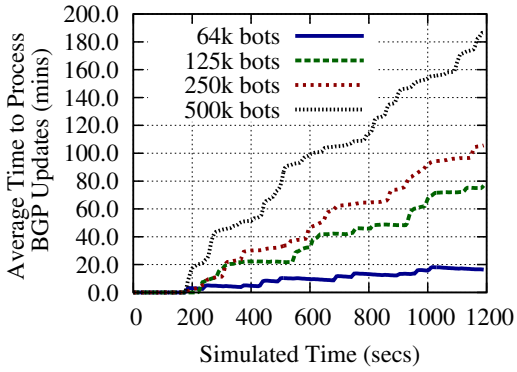


Figure 4.5: The average time to process a BGP update for core ASes under attack by botnets of various sizes. Attack traffic starts at time 0, the first link failures occur at time 180.

We term the average delay between when a BGP update arrives and when it completes being processed the time-to-process or TTP. The TTP for core ASes under attack by various sizes of botnets is graphed in Figure 4.5. CXPST successfully triggers the first BGP session failures 180 seconds into the attack. From this point onward the average TTP for updates arriving to core ASes increases dramatically. For example, in the case of a $250,000$ node attacker, after 10 minutes of attack the backlog of updates is large enough to delay processing for roughly 45 minutes. Once 20 minutes of attack time have passed the wait has increased by an additional hour, to just over 100 minutes. The reason for this constant increase in TTP was discussed in Section 4.1.1. Routers under this amount of computational load are resource exhausted, and can only recover if they

are receiving update messages at a low rate. However, updates are nearly constantly arriving as a result of CXPST. This means that the affected routers are never given a chance to recover.

Anecdotal evidence suggests that routers placed in resource constrained states behave unstably [44]. It is not outside the realm of possibility that, when confronted with update queues thousands of messages long and processing delays measured in minutes rather then microseconds, that routers will exhibit undefined behavior. This undefined behavior adds a new dynamic to the system. We leave study of this for future work.

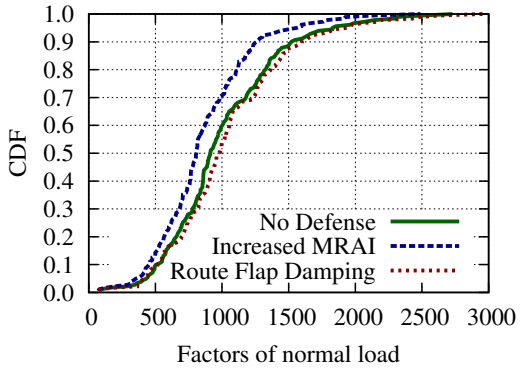
4.4 Toward Defenses

Given the potential consequences of an adversary carrying out CXPST, building a defense against it is of paramount importance. Since CXPST is route flapping on a grand scale, mechanisms that mitigate the damage done by route flapping might prove successful. We will examine these technologies and demonstrate that they do not have an effect on CXPST. We then focus on stopping CXPST before it has an opportunity to generate update messages rather then trying to change BGP's tendency to broadcast updates globally. We do this by proposing a simple configuration based solution to prevent Zhang et al.'s attack. Our solution has the advantage of being easily deployable and effective, even if only partially deployed.

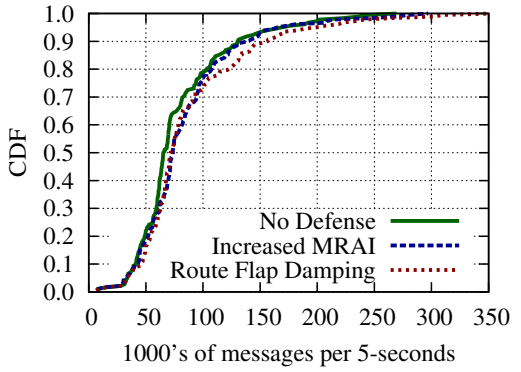
4.4.1 Deployed Defensive Measures

As discussed in Section 4.1.1 there are a handful of currently deployed mechanisms to reduce the number of updates generated by route flapping. We ran a set of simulations using a $250,000$-node attacker in an effort to evaluate the effect of these defenses on CXPST.

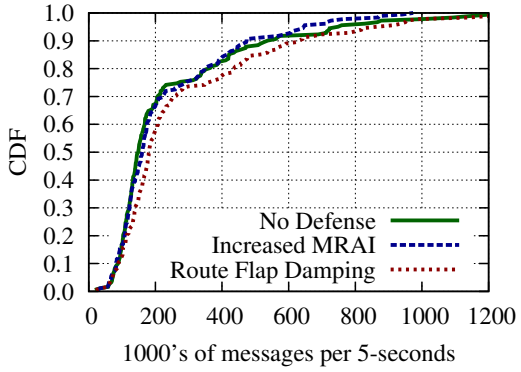
In our experiments, BGP Graceful Restart [28] did not have a significant effect on the behavior of the network. BGP graceful restart is meant to provide a grace period to the data plane when a BGP session fails. Because our attack traffic travels on the data plane, it benefits from this grace period, allowing CXPST to continue stressing the link until the grace period expires. When this happens the resulting situation is the same as the one that occurs when BGP Graceful Restart is not used.



(a) Median Load with Route Flapping



(b) 75th Percentile with Route Flapping



(c) 90th Percentile with Route Flapping

Figure 4.6: Median router load of targeted routers using route flapping defensive measures as a factor of normal load (4.6a), 75th percentile (4.6b), and 90th percentile (4.6c) of message loads experienced by routers using defensive measures, while under attack by 250,000 bots, measured in BGP updates seen in 5-second windows.

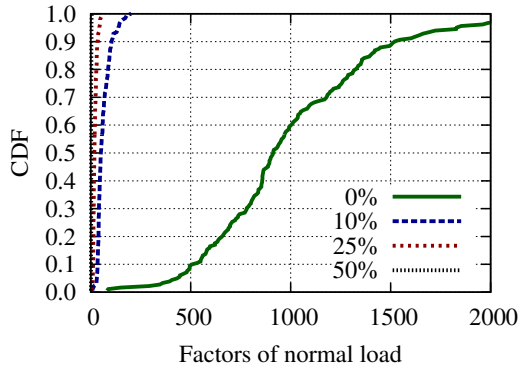
The other two defensive measures had nearly as limited an effect. A comparison of CXPST, CXPST run in a system with increased minimum router advertisement intervals, and CXPST run in a system with globally deployed route flap damping can be seen in Figure 4.6. As can be seen in these graphs, the defenses, as predicted in section 4.2.5, do not have significant impact on CXPST.

4.4.2 Stopping Session Failure

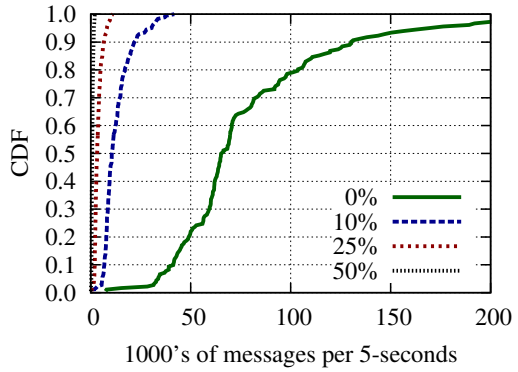
Instead of attempting to limit the scale and number of updates that result from CXPST, our proposed defense focuses on stopping CXPST before it can generate updates. Our defense against CXPST is simple, remove the mechanism that allows Zhang et al.'s attack to function. Sadly, accomplishing this is easier said then done. Creating a differentiated service class for control plane traffic, if done correctly, could solve this issue. The issue with this is that existing routers are incapable of correctly providing this "perfect service" class.

One simplistic way to stop the ZMW attack is disabling hold timer functionality in routers, something easily achievable by setting the timer to an exceedingly large value. By doing this, BGP sessions will not be terminated by high amounts of data plane traffic. This can be achieved with a simple change to configuration files, making its cost negligible, but it is unclear if modern network monitoring is nimble enough to correctly assume the responsibilities of hold timers. Nevertheless this solution is illustrative of any mechanism that protects BGP session failure from data traffic.

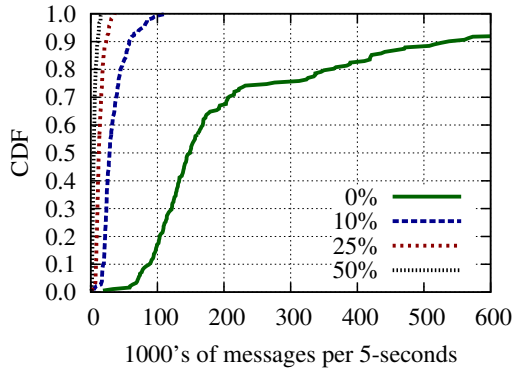
It is unlikely that any solution to the ZMW would be globally deployed. For example, not all network operators possess the same level of network monitoring, and most will be unwilling to remove hold timers. In order to test if our defense is incrementally deployable, we simulated a $250,000$-node botnet running CXPST against a network that had fractional deployment of our solution. We selected the largest ASes by degree to implement our solution. The results of these tests can be seen in Figure 4.7 and Figure 4.8. We discovered that if as few as 10% of the ASes implemented our solution, it would dramatically reduce the impact of CXPST. In fact, our simulations suggest that a 50% deployment would be sufficient to stop CXPST completely.



(a)



(b)



(c)

Figure 4.7: Router loads of targeted routers under attack by a 250,000 node botnet when various fractions of routers have disabled hold timers.

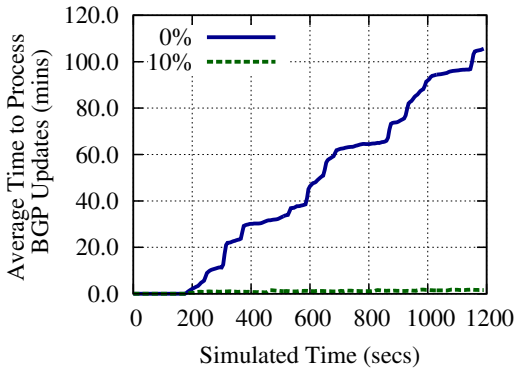


Figure 4.8: The average time to process a BGP update for core ASes when hold timers are removed for 10% of routers. The delay with no defense deployed is provided as a reference. Attack traffic from the $250,000$ bots starts at time 0, the first link failures occur at time 180.

Chapter 5

Using Resource Consumption To Trigger Router Failure At A Distance

While routers function well under *normal* conditions, there is one obvious question: What happens if one router forces another to operate under *abnormal* conditions? In this chapter, we will demonstrate that an adversary in control of a router can cause an *arbitrary* honest router on the Internet to fail, even if the adversary is *not directly connected to the victim*. We present the results of a collection of experiments on hardware and software routers running the Border Gateway Protocol (BGP) to illustrate three key points. First, that unexpected but *perfectly legal* BGP messages can place routers into unstable states with troubling ease. Second, that an adversary can implement attacks using these messages to disrupt the function of victim routers in arbitrary locations in the network. And third, that modern best practices do not blunt the force of these attacks sufficiently.

Through experimentation on hardware and software routers, we examined what happens when modern routers find themselves without free memory or computational resources. In all cases, we found that routers fail to handle these scenarios gracefully. We witnessed a variety of failure modes, ranging from severe performance degradation to the unrecoverable failure of all active routing sessions. Given the negative impact of these unstable states, one might be tempted to believe that placing a router in such a state is difficult. Sadly, it turns out that the opposite is true. We found it relatively easy to send BGP messages to a router that would directly place it into one of these states. We focused on exploring corner cases that are unlikely to be found "in the wild" but are still perfectly valid in the eyes of BGP. We will examine examples of these messages later in the chapter.

While it is unlikely a router would see any of these messages normally, they are easily generated by an adversary. By utilizing these BGP messages, an adversary who controls a BGP speaker is capable of launching powerful attacks against other routers. We will show how our adversary can manipulate honest routers into propagating these malicious BGP messages across the network, allowing our adversary to attack routers not directly connected to himself. In addition, by triggering loop detection mechanisms, the attacker can contain attack updates so that they are only seen by routers on paths between the malicious router and victim.

It might be comforting to assume that deployed routers could be hardened to these attacks via proper configuration. We demonstrate that the commonly accepted best

practices would do little to slow these attacks. We examine four options in detail: prefix filtering, prefix aggregation, prefix limits, and AS path length limits. In each case, we use observations based on the contents of real world routing tables to reason about both the extent to which these best practices are used and the degree to which these practices could prevent our attacks.

5.1 Crashing Routers

In this section we will outline the mechanisms an adversary in control of a BGP router can use to disrupt the functionality of victim routers. We will examine how an adversary can consume all of a target's free memory, causing the target to crash. As covered in Section 2.2, routers have two different types of memory, small amounts of high speed memory for line card operation, and larger amounts of general purpose memory for control plane operation. We are interested in the latter, consequently for the rest of the paper when we refer to *memory* we mean general purpose memory. Previous research by Chang et al [44] examined what happens when a router runs out of free memory. They found that when a router exhausts its free memory, the BGP process crashes, causing the failure of all BGP sessions and the halting of data plane operation.

Our observations in this and later sections come from experiments run on hardware and software routers. The hardware router we had access to was a CISCO 7603 series router. It is important to note that in this work we are experimenting with the behavior of a router's *software*, not its hardware. Our 7603 is an acceptable test router since it runs the same version of IOS deployed on many larger and more powerful CISCO routers. For a software router we selected the Quagga suite. In order to ensure isolation, our software routers were run on Qemu virtual machines running Linux. Our experiments in this section were done with a simple topology where attacking routers, in this case BGP injectors, were directly connected to the victim router. In Section 5.3 we will expand to more complicated topologies to explore how our attacks function on distant targets.

5.1.1 Available Router Memory

The immediate question that springs to mind is: how difficult is it to force a router to exhaust its supply of memory? Clearly this is dependent on how much free memory a

router has. The amount of total memory varies widely based on exactly what model of router is used and where in the network it is deployed. As discussed in Section 2.2, routers commonly have between 256 MB to 4 GB of *total* memory. However, what we are interested in is the *free* memory. The main demand placed on a BGP router's memory comes from storage of the Routing Information Bases, RIBs, which are tables of known valid routes, currently used routes, and currently advertised routes. It is easy to see now how a router's position in the network will alter the demands on its memory. If a router is located in the dense core of the Internet, where it has a large number of peers, the majority of which are advertising a global BGP table, it will have higher memory usage compared to a router that exists on the fringes of the Internet, where it might only receive one or two global BGP tables along with a collection of very small tables.

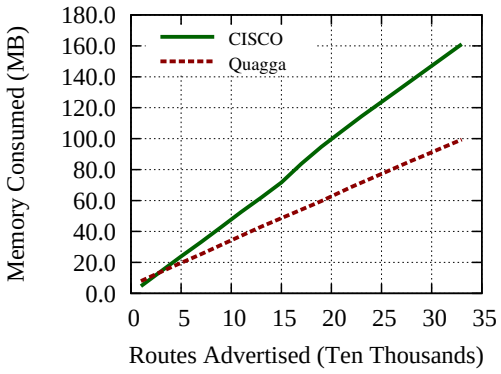


Figure 5.1: Measured memory consumption of BGP process as a function of the number of real world routes advertised to it.

Because of this diversity, building an exact model of router memory usage is impossible. We can build a rough estimate by examining both how much memory a single global routing table requires and how many of those tables a router could in theory receive. The first quantity can be measured directly. We collected global routing tables from April of 2012 via RouteViews [17] and performed a series of experiments involving our routers. We advertised global routing tables to both types of router and measured their memory usage. The plot of measured memory usage versus the total number of prefixes can be seen in Figure 5.1. We then examined a sample of representative routers, noting how many line cards the routers can have, presenting an upper bound on the

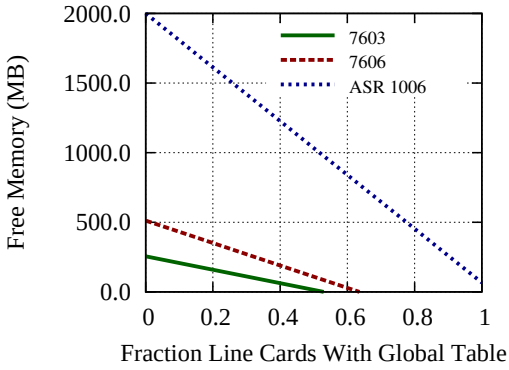


Figure 5.2: An estimate of the upper bound of free memory in various models of CISCO router as a function of the number of line cards receiving full global routing tables from peers.

number of full tables a router could receive [1] , and their total memory. We can combine this data and the results in Figure 5.1 to build an upper bound on the amount of free memory a router could have as a function of the fraction of its line cards receiving a global table from a neighbor, shown in Figure 5.2. While this figure does not give us a definitive target for memory consumption, it does give us an idea about the state of free router memory. Smaller routers, like the 7603, will more than likely have little to no free memory available, while larger, more powerful routers will not have free memory in abundance, likely no more than 1 GB.

5.1.2 Consuming Memory via Updates

We can now consider how an adversary would consume the memory of a target router. Our basic approach is to send BGP advertisements to the victim router, which will in turn store those advertised routes in its RIB. The challenge for the adversary is to craft BGP routes that take up as much space as possible. It is important to note that the routes the adversary sends need not be considered the best by the victim, they need only be valid. We set out to establish what is the maximum amount of memory an adversary could consume per BGP route accepted. Routes crafted using various strategies were advertised to the target router, and its resulting memory load measured.

[1] Routers could in theory have more BGP sessions than they have line cards, via multi-hop BGP sessions, however these are not commonly used in practice.

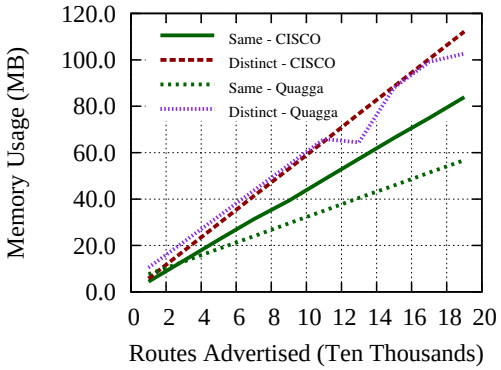


Figure 5.3: Measured memory consumption of BGP process observed byrepeating the Chang et al experiments compared to distinct routes.

An starting point for building an update construction strategy would be to revisit the experiments done by Chang et al [44], where injectors simply advertised successive IP blocks with a single hop AS path. We re-ran their experiments with our CISCO router, the results of which can be seen in Figure 5.3. We estimate that it would take on the order of 2 million routes in order to exhaust the free memory on modern routers using their methods, but can an adversary do better?

The first thing to notice is that all of the AS paths being advertised in the first set of experiments are identical, simply the ASN of the advertising router. In an effort to shrink the memory footprint of RIBs, routers only store identical AS paths once. In fact, if one compares the memory usage in Figure 5.3, where all AS paths are the same, to the memory usage from storing real world routes in Figure 5.1, where there is some degree of path distinctness, one can quickly see that the small amount of path distinctness results in increased memory load. Therefore, if the adversary can ensure all of the routes being advertised have distinct paths, we should see a larger increase in memory usage. Figure 5.3 shows memory consumption when routes have distinct AS paths compared to routes with identical AS paths. This increase in memory load of 33% is a start, but clearly the adversary still needs to consume more memory per update for the attack to become realistic.

The next observation is that we can make the AS paths longer, forcing the router to spend more memory storing them. While the BGP RFC allows AS level paths of

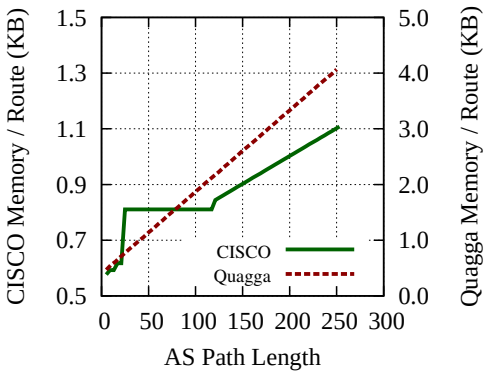


Figure 5.4: Per update memory usage as a function of path length for Quagga and CISCO routers for both unique and identical sets of paths. Note how Quagga's memory allocation is always a function of the path length, while CISCO allocates fix size blocks for all paths between 24 and 108 ASes in length.

any length so long as they are packaged in the update message correctly, software bugs related to long AS paths and the practice of limiting AS paths (discussed in Section 5.4.5) constrain our adversary's ability to create *arbitrarily* long paths. Of course, paths longer than the single hop used by Chang et al are still possible. We ran another series of injections using distinct AS paths of varying lengths. The per path memory consumption as a function of path length can be seen in Figure 5.4. While the Quagga router allocates memory proportionally based on AS path length, we can see that the CISCO router instead allocates memory in a fixed size block for AS paths longer than 21 and switches to a proportional allocation only for paths longer than 120. This means that by advertising AS paths of length 22, the adversary can consume the same amount of memory he could by advertising paths of length 120. This is important, as AS paths of length 22 are smaller than limits imposed by current best practices, something we will cover in more detail in Section 5.4.5.

Our last observation is that our simple update, even with an above normal size path, only takes up a small fraction of the total available bytes in a BGP update message. The adversary can fill the remainder of that space with Community Attributes [9] in an effort to consume more memory. Community Attributes are well known optional transitive attributes which allow operators to specify arbitrary path qualities. The impact these attributes have on memory usage are highly similar to AS paths: each unique community

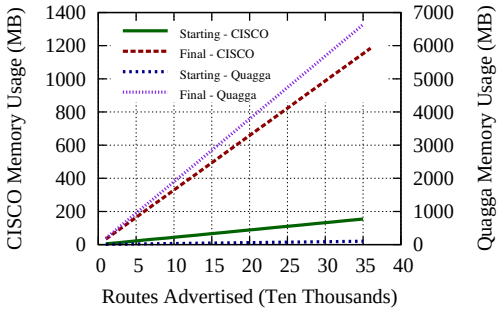


Figure 5.5: The measured memory consumption when applying path distinctness, increased AS path length, and distinct community attributes compared to Chang et al's attack.

attribute needs to be stored in a receiving router's RIB, and increasing the number of community attributes increases memory consumption. We repeated our route injection experiments, padding the update messages with unique community attributes. A plot of memory usage as a function of accepted routes can be seen in Figure 5.5. Memory values beyond the capacity of our CISCO router, are extrapolated. We can see that the combination of distinct AS paths, increased modestly in length, and surrounded with unique community attributes increases the per route memory consumption by a factor of 7.48. The change is dramatic. Instead of needing more than 2.2 million routes to consume 1 GB of memory, 300 thousand routes can accomplish this task, a fraction of the size of today's global routing table. In fact the *total* memory of our CISCO router could be consumed with 76 thousand routes.

5.1.3 Exhaustion Through I/O

An alternative tactic for exhausting a router's free memory is by increasing the demands from its I/O buffers. We observed crashes that resulted from running out of memory in our victim router when its *neighbors* became CPU starved. To understand why this occurs, we must take a look at what happens when the rate of incoming updates to a router exceeds its computational capacity. In this case the receiving router will have to buffer the unprocessed updates. We found that both our Quagga and CISCO router will only buffer a fixed number of BGP messages. When those limits have been reached, the BGP process will stop fetching packets from the operating system's buffers. Network

buffers are of fixed size as well—when the receiving router's network buffer is full, it will send TCP ZERO WINDOW messages to the advertising router, preventing new packets from being placed on the wire. New packets are then buffered in the *sender's* network buffers. When those fill, the updates are buffered inside the advertising router's BGP process. These buffers are *unbounded* in size. We term this behavior *back pressure*.

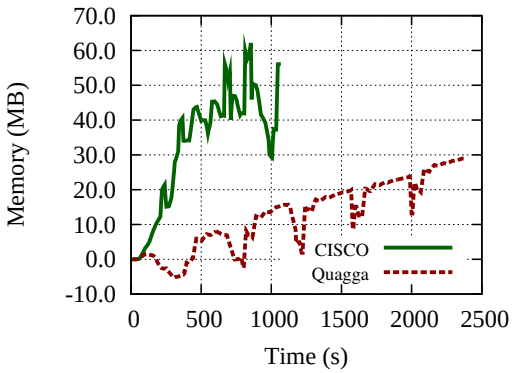


Figure 5.6: The increase in memory load for a Quagga and CISCO router when advertising to a CPU starved peer compared to a normal peer, demonstrating exhaustion of memory via I/O backlog.

We performed a simple experiment to illustrate back pressure involving three routers: an injector, a victim router, and a potentially computationally starved peer. The injector was directly connected to the victim router, as was the CPU strained router. All routers started with empty tables, and the injector would proceed to advertise routes taken from RouteViews to the victim router, which would in turn attempt to advertise them to its other peer. Runs were performed when the third peer was both CPU starved and when it had sufficient computational resources. The *difference* in memory loads of the victim router when its peer was CPU strained versus when it was not can be seen in Figure 5.6. The CISCO router experienced an increase in memory consumption of more than 40 MB, with spikes over 60 MB. Quagga saw an increase of 30 MB by the end of our experiment. It is important to note that this was for a single CPU strained peer, and that these added memory costs are *per peer*, meaning that routers with more peers are more susceptible to this attack.

This increase in memory usage was not the strangest behavior that resulted from update back pressure. On the CISCO router, we noticed that if the amount of back

pressure was large enough, the processes controlling BGP I/O started to fail. Specifically, it ceased interacting correctly with the peers responsible for the back pressure. The memory-starved router ceased attempting to send BGP related packets to these peers. We assumed tearing down the BGP session, which would result in a new TCP session, would solve this I/O issue. It did not. While the back pressure causing peers could complete a TCP handshake, no response to their BGP OPEN message came from the CISCO router. This I/O issue was limited to BGP, however, as we could initiate a telnet console with the CISCO router from the Linux box hosting the troubled peer. This problem was only fixed when the CISCO router was restarted.

Of course in order for back pressure to exist a router's neighbors need to be in a CPU starved state. This can occur for a variety of reasons. The most obvious one is as a result of nearby router failures. As discussed in Chapter 4 router failures push the network out of convergence, forcing routers to spend processing power recomputing the best paths to large swaths of the IP address space. This is not the only way however, as we will discuss in Section 5.2, an adversary can exhaust the CPU of routers in a targeted manner using BGP updates.

5.2 CPU Exhaustion

In our experimention we came across two different ways to exhaust the CPU resources of a router. The first is obvious: we can force the targeted router to process large numbers of updates. While an adversary could simply push as many updates into the network as fast as he could, this attack would be trivially thwarted when upstream peers simply ignore updates from the adversary. Instead, we look at how an adversary can coerce honest routers into sending large numbers of BGP advertisements to each other. The adversary accomplishes this by crafting single updates that, when propagated through the network, will cause the targeted router to destroy one or more active BGP sessions with other honest peers. The second method for exhausting a router's CPU is algorithmic DoS. We found a pair of these attacks that worked on our test routers, one for Quagga and one for CISCO. In both the attacks against BGP sessions and the algorithmic DoS attacks, we utilize valid advertisement messages that are designed to take advantage of slow or buggy code designed to handle odd "edge cases" found in BGP.

5.2.1 BGP Notifications

In our examination of router behavior we found a large number of BGP messages that are valid according to the RFC, but cause the recipient to send a BGP NOTIFICATION. Receiving a BGP NOTIFICATION causes the failure of the the corresponding BGP session, which will result in routers being pulled out of convergence, a state requiring the processing of large numbers of BGP advertisements to fix. This problem is compounded by the fact that since all of these messages are valid, they can be generated by honest routers.

One set of these valid but problematic messages revolves around AS path length. BGP breaks paths up into segments which, as defined by the RFC, can be at most 255 ASes long. If a router is passed a segment that is longer than 255 ASes that path will result in a BGP NOTIFICATION. In order to send a path longer than 255 ASes, the path needs to be split into two sequences. For example, to send a path of 256 ASes the first segment could have a length of 1 and the second a length of 255. This path is perfectly legal, but both the CISCO and Quagga routers respond as though it is invalid. In the case of Quagga, we can look at the source code to discover that when Quagga encounters a pair of adjacent sequences in an AS path, it merges them without checking their sizes first.

The CISCO router handles even more path length cases incorrectly compared to Quagga. Unlike Quagga, it responds to *any* path of length 255 or larger with a BGP NOTIFICATION. We noticed that the CISCO router would often claim routes were longer than 255 ASes when they were actually shorter. We noticed this behavior when experimenting with AS paths that contained several AS segments. We established through test cases that if the sum of the actual AS path length plus half the number of segments in the path was greater than 255, then it would send a BGP NOTIFICATION to the peer that advertised the route, claiming that the route was invalid because its length was greater than 255. For example, a valid route comprised of 25 AS segments of length 10 each would be considered to have a total length longer than 255 AS segments, even though its actual total length is 250. This behavior is consistent with the CISCO router computing AS path length by taking the total byte size of the AS path portion of the update message and simply dividing by two, the expected wire size of an ASN, instead of reading from the packet the actual segment sizes.

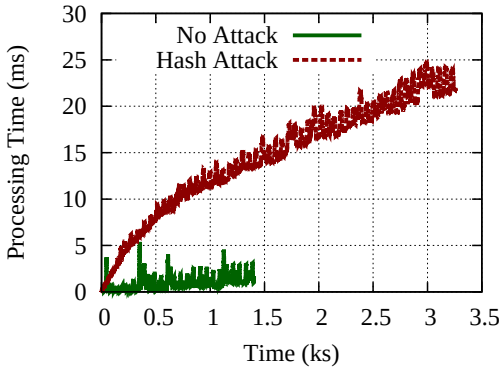


Figure 5.7: Comparison of the time required to process a set of paths designed to collide into one hash bucket versus a set of random paths.

This assumption about wire size was also an issue when the CISCO and Quagga routers interacted. Our Quagga routers attempted to write ASNs to the wire as 4 bytes rather than the 2 bytes seen from the CISCO router. This presented several issues. First, the CISCO interpreted AS paths from Quagga routers as longer than they actually were, with all of the consequences we have outlined previously. Second, if updates were very large in size, this expansion from 2 to 4 bytes resulted in update messages going over their maximum size of 4,096 bytes. The real problem here is the asymmetric behavior of Quagga and CISCO routers. A collection of CISCO routers could pass between each other update messages near the 4,096 byte limit without error. When those messages reach a Quagga router, the AS path size will increase as the Quagga router expands the AS size. If that expansion pushes the update messages over the 4,096 byte limit, they will be responded to with a BGP NOTIFICATION.

5.2.2 Hash Attack

Quagga stores AS paths inside a hash map, allowing for rapid access. In the Quagga hash map implementation, the map is of fixed size, in this case 2^{15}, and the hashing function is predictable. This is acceptable so long as the assumption that AS paths will be spread evenly over all of the buckets holds. However, an adversary can violate this assumption. By computing a large number of AS paths that hash to the same value and advertising them to a router, we can increase the amount of time route processing

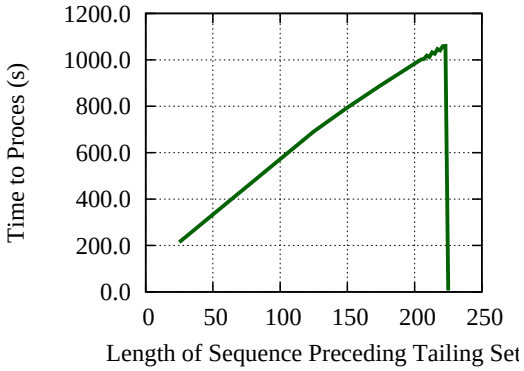


Figure 5.8: The time required for the CISCO router to process 8,000 routes with fixed length unique paths that ended in an AS set as a function of the size of the AS sequence preceding the tailing set. The far right hand value is for paths without a tail set.

takes. This class of attack, first proposed by Crosby and Wallach [45], exploits the fact that while inserting n elements into a hash map would normally take $O(n)$ time, if each element hashes to the same value it will take $O(n^2)$ time. Plots of the time to process updates with colliding AS paths compared to random AS paths can be seen in Figure 5.7.

Path Sequences and Sets

AS paths are made up of one or more segments. These segments come in two different flavors: sequences and sets. An AS sequence denotes an ordered list of the exact ASes the path utilizes. On the other hand, an AS set contains a collections of ASes and asserts that the path will utilize one or more of them, without giving further details. The overwhelming majority of paths on the Internet are formed by a single AS path sequence. In nearly all cases, AS paths end with a sequence, but there is nothing that prevents a path from ending with a set. While our Quagga router handled this scenario normally, the hardware router experienced issues.

The CISCO router took several orders of magnitude more time to process routes that ended in an AS set compared to routes that ended normally. In a series of experiments, we injected paths with variable lengths of trailing sets while holding the overall AS path length constant. The results of these runs can be seen in Figure 5.8. In these runs we injected 8,000 routes with unique paths of length 225. The paths with no trailing set

took 6 seconds to be processed, while the paths that added an AS set of length 2 took 1,060 seconds. Note that the length of time to process is not dependent on the size of the AS set, but rather on the size of the AS segment that precedes it. It is also interesting that this behavior only exists if the path ends in a set: if the set of paths that took 1,060 seconds is modified to have an AS segment of size 1 after the set, then the CISCO router again takes 6 seconds.

5.3 Attacking Distant Routers

The attacks of Section 5.1 provide a peek into how a malicious router can force a victim into a non-functional state via legitimate BGP messages. However, one might feel skeptical about these examples, as they were tested with an adversary that is directly connected to its victim. In this section, we will provide examples of how an adversary in control of a router could force other routers *at arbitrary locations in the topology* into an unstable state. We will lay out how our adversary can launch this attack by convincing honest routers to forward these malicious updates through the network to the victim while at the same time minimizing his direct impact on other routers in the system.

5.3.1 Threat Model

Our threat model focuses on legitimate BGP speakers in transit ASes [2] that have become malicious. These adversaries are the result of either an autonomous system electing to act in an adversarial manner or an outside entity compromising one or more BGP routers. We focus on transit ASes since stub ASes have very limited abilities within the BGP network. Our chosen threat model gives our adversary two key capabilities.

First, the adversary can send BGP messages to other routers. The malicious router cannot simply send arbitrary messages to *any* router, however; it can only directly send BGP messages to its legitimate peers. This is an issue for our attacker, as his previously stated goal is to disrupt *arbitrary* routers or BGP sessions, not simply those he is directly connected to. In order to have malicious update messages reach arbitrary routers, our adversary will need to convince honest peers to propagate those updates in

[2] By transit AS, we mean any AS that has other ASes as customers.

such a manner that the intended victims receive them. We will cover how our adversary does that in Section 5.3.2.

The second ability our adversary has is the capacity to act in a non-standard, or even protocol non-compliant manner. Our adversary can, for example, locally ignore paths with loops, use non-standard path selection, not apply best practices, and advertise paths in a manner that does not conform to Valley Free Routing. However, our adversary again runs into the issue that *only* he can act in this way; honest nodes will act normally and can use best practices. We will cover how these attacks work in relation to best practices in Section 5.4.

5.3.2 Propagating Malicious Updates

In our threat model the adversary only has the capacity to send update messages directly to his legitimate peers. In order to get malicious updates to targeted routers, the adversary will need to convince routers that lie on paths between him and his victim to forward the updates. Honest routers only re-advertise the routes they consider "best". This is an issue for our adversary because, as we have seen in Section 5.1, some of the malicious updates will have a longer than average AS path length. Because AS path length is one of the key path selection metrics, this will make it less likely that the malicious updates will be considered best if there is an alternative.

If our attacker could advertise IP blocks that have no competing paths, the malicious routes would be the best by default. To do this, our attacker will take advantage of the fact that BGP considers more specific IP prefixes to be distinct. For example, BGP considers the IP block `123.101.0.0/16` to be distinct from `123.101.128.0/17` and `123.101.0.0/17`. Since these blocks are considered distinct, path selection for `123.101.128.0/17` will be done separately from `123.101.128.0/16`. Our adversary simply couples his malicious advertisements to highly specific IP blocks (e.g. `123.101.128.0/24`) for which there are not pre-existing routes. Due to this forced de-aggregation, his malicious updates will have no competition and will be the best. We discuss how this tactic interacts with best practices such as aggregation and prefix length filtering in Section 5.4.

5.3.3 Building Attack Flows

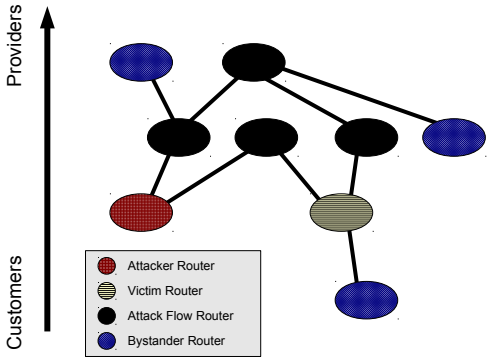
The adversary cannot blindly send these "best paths" out to all of his peers in the hope that they eventually reach his target. Our adversary is going to take advantage of the fact that honest routers operate in a predictable manner in order to construct "flows" of updates from himself to the victim. When determining whether to propagate a best path to its neighbors, a router takes into account the customer/provider relationships it has both with its neighboring routers and with the route's next hop. A well known set of policies called Valley Free Routing [46] are applied based on these relationships. Valley Free Routing states that a path will be advertised if and only if: a) the party being advertised to is the AS's customer or b) the route was learned from a customer. While the AS relationships are technically private information, a large amount of work has been done to infer them. By building a topology based on a data set of these relationships between ASes [17] and applying Valley Free Routing policies to this topology, a model for how the malicious routes will travel through the network can be built. Central to this construction is the concept of a *customer cone*. An AS's customer cone is the set of all of its customers, plus all of its customer's customers, and so on.

With this model of path propagation in mind, the adversary can construct a directed graph based on Internet topology and inferred AS relationships. The adversary starts with an edge-less AS graph. The adversary then adds a directed edge between his AS and every AS he is directly connected to, this represents his ability to send advertisements to any peer he is directly connected to. For each of the ASes added to the connected component containing the adversary that are part of the adversary's customer cone, an edge is added from that node to its customers. This represents the "A" clause of Valley Free Routing. If instead the AS added to the connected component is the adversary's provider, an edge is added from that AS to all of its customers, peers, and providers, the "B" clause of Valley Free Routing. If a path exists from the adversary to the victim, that means that there is some neighbor of the adversary which, if the adversary sends an update, will start up a chain of advertisements that will end with the update reaching the victim. We call such a path through this directed graph an *attack flow*. In a densely connected graph such as the Internet, there will typically be multiple paths, which will allow the adversary to load balance his malicious updates. We will examine the prevalence of attack flows in Section 5.3.4.

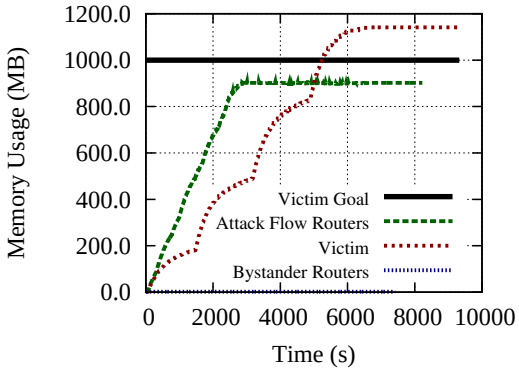
Once attack flows are found, it might be in the attacker's interest to contain updates to only their assigned flow. The adversary must prevent routers that are next to the attack flows from accepting the malicious routes. To do this, our attacker can use loop detection to his advantage. In BGP, loop detection is achieved by scanning the AS path for the router's ASN. If the router detects itself in the path, it considers the route in-feasible, neither storing it in memory nor propagating it. It is important to note that when loop detection is triggered it does *not* result in a BGP NOTIFICATION. The first step for our adversary is to define the "borders" of each attack flow. These are all of the nodes in the directed graph which are *not* part of the attack flow, but have an edge leading from a node in the attack flow to themselves, we call these nodes *bystander nodes*. These are the nodes that will see the malicious updates, but have nothing to do with the actual propagation of the updates to the victim. He then ensures that the ASNs of all neighbors of the attack flow are included inside the fabricated AS path of any update utilizing that flow. Since the neighbors of the flow will not propagate the malicious updates, the routers behind those neighbors will never see the updates.

5.3.4 Experimental Observations

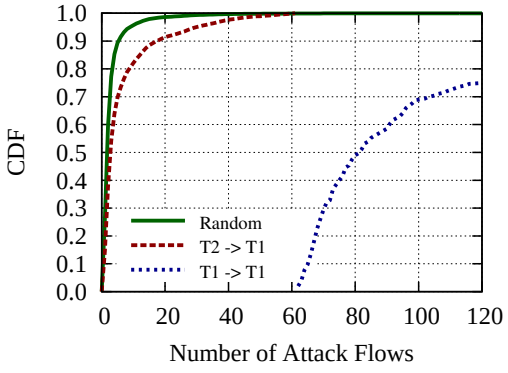
In order to validate that our attack works, we utilized our software router test-bed from Section 5.1. Our topology was a small subset of the AS level topology of the Internet built by doing an expansion from a node chosen at random. An example topology including AS relationships can be seen in Figure 5.9a. We launched the attack we have just described from random attackers to random victims and monitored the memory of routers in the test-bed. We set a goal, based on our findings in Section 5.1.1, of 1 GB of additional memory consumption. The routers in the test-bed fall into one of three different groups. First, there are routers that are on the attack flows from the adversary to the victim. Those routers will see increased memory utilization, but *not* enough to push them over the 1 GB threshold. Second, there is the victim, who should see memory consumption *over* the 1 GB threshold if the attack is successful. The last group of routers are those directly connected to the victim or an attack flow, but not actually part of the flow. Our loop detection technique should cause them to see *no* additional memory load.



(a) Example Attack Topology



(b) Example Memory Loads During Attack



(c) Available Attack Flows

Figure 5.9: A graphical representation of one topology configuration used for our experiments is shown in Figure 5.9a. Figure 5.9b is a trace of memory loads for BGP processes of various classes of routers during an update based memory exhaustion attack. A CDF of the number of attack flows existing between various tiers of transit routers based on AS level topology can be seen in Figure 5.9c.

A collection of memory traces from an example run can be seen in Figure 5.9b. In this run two attack flows were used by the attacker to push malicious updates. The injection of malicious routes to directly connected neighbors starts at time 0 and completes at roughly time 2900. As can be seen, the attack flow routers show an increase in memory that ends with them having less than the 1 GB barrier, exactly what we expected to see. The victim router's memory load lags behind the average memory load of the attack flow routers, a result of it being the last router in the chain that the updates propagate to. At time 5200 we see the victim router cross the 1 GB threshold. In our case the router did not crash as the virtual machine had added memory to handle this load. The last set of routers, those protected by loop detection, showed zero increased memory load, as all of the incoming routes were discarded as loops.

Counting Attack Flows.

Another interesting question to ask is how many attack flows an adversary is likely to have available for a given victim. To answer this question, we examined the AS level topology (considering only those ASes that meet the threat model discussed in Section 5.3.1) and calculated the attack flows between each pair of ASes, counting the number of flows. The results of this calculation appear in Figure 5.9c.

As the figure shows, in the worst case a random adversary has a 73.5% chance of possessing multiple attack flows to a random victim; almost half of all pairs have at least 3 attack flows. Furthermore, adversaries in the "transit core" of the Internet are even more advantageously situated: for example, around 10% of tier 2 routers can access 20 or more attack flows when targeting a tier 1 router; and if an adversary is in control of a tier 1 router, at worst he has roughly 60 attack flows open to him, and on average far more. This supports our contention that our attacks can indirectly target arbitrary routers in the AS topology.

Hash Attacks on Real Topologies

In addition to the memory exhaustion attack, we also validated the hash attack from Section 5.2.2 on our topology. Our adversary in this attack computes hash collisions for the target router in the manner previously covered with one exception, the ASes in the attack flow need to be taken into account. Hash collisions are computed by inverting

the AS path taken by the attack flow, and using this as the start of the route. ASNs are then appended to the back of the route such that the entire route is a hash collision when it reaches the victim router, but not before. A plot of the time to process incoming routes for the target router along with a representative attack flow router can be seen in Figure 5.10.

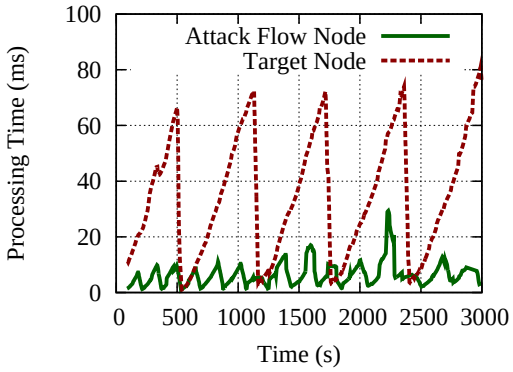


Figure 5.10: Trace of time required to process incoming BGP updates for two routers, one a victim, the other a router on an attack flow to the victim during a hash attack.

5.4 Defeating Best Practices

In this section, we examine how various "best practices" interact with the proposed attack of Section 5.3. We focus on the following practices: prefix length limits, prefix filters, prefix aggregation, prefix count limits, and AS path limits. We will show that each of these fails to disrupt the adversary's actions to any sizeable extent. We also consider how a global deployment of BGPSec [13] would impact our adversary.

5.4.1 Prefix Length Limits

One commonly applied best practice is to drop updates for highly specific prefix blocks. Filtering in this manner is done in an effort to control the size of routing tables. This policy is an issue for our attacker because, as discussed in Section 5.3.2, our adversary relies on advertising very specific prefix blocks which do not have pre-existing paths. Two questions are raised because of this practice.

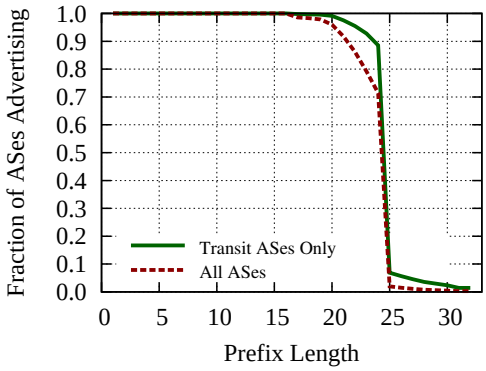


Figure 5.11: he fraction of ASes observed view RouteViews advertising prefixes of a given length.

First, how specific of a prefix can our adversary advertise without it being filtered? To answer this, we examined what length of prefixes we can *actively observe* being forwarded by various Autonomous Systems based on RouteViews data, the results of which can be seen in Figure 5.11. What we found was straightforward: 88.5% of transit ASes forwarded prefixes that were /24s or shorter, while 6.8% forwarded prefixes longer than this. Thus, in the majority of cases, our adversary can advertise routes containing a /24 or shorter successfully.

This leads us to our next question: Given that we can advertise no more specific a prefix than a /24, can our adversary find enough un-advertised prefixes to complete his attacks? This can be answered with a quick back of the envelope calculation. There are approximately 1.6×10^7 prefix blocks of length /24, and of those 98% correspond to routable IPs (the other 2% are un-routable bogons [47]). The current size of the full Internet routing table is roughly 4×10^5 prefixes [48], meaning that if all IP blocks advertised were /24s then there would still be over 1.5×10^7 /24s un-advertised. Clearly this means that our adversary can find a sufficient number of un-advertised /24 blocks to utilize for his attack.

5.4.2 Prefix Filtering

Modern routers have the ability to filter incoming updates based on a combination of the CIDR being advertised and the AS sending the update. This capability, if widely

and correctly deployed could have an impact on our adversary's strategy as outline in Section 5.3. Sadly, it is *not* a foregone conclusion that prefix filtering is deployed in such a manner. Prefix filtering must be done manually, and changes in filters requires a reconfiguration of routers. Additionally, establishing exactly what blocks of IP addresses an AS should or should not advertise is a non-trivial task. Measuring exactly how many ASes deploy prefix filtering correctly would require operators to share confidential information, specifically the active configurations of their routers. However, we can gain an intuition as to how widely and correctly prefix filters are deployed by examining several historical incidents of prefix leaks and hijackings [49, 2, 12] were only been possible because prefix filtering is not correctly and widely used. As an example, in 2010 China Telecom accidently sent advertisements for 50 thousand blocks of IP addresses that it did not own. If prefix filters had been in place, this incident would have never had an impact outside the misconfigured routers.

5.4.3 Prefix Aggregation

Tied closely with the subject of filtering long prefixes is the concept of prefix aggregation. Upstream routers have the ability to aggregate multiple advertisements from downstream peers into a single, less specific, advertisement which they pass on to their peers. This again presents an issue for our adversary, as aggregation could cause his attack updates to be merged into a small number of aggregated routes. However, this issue is actually a non-factor for our adversary for several reasons.

Aggregation must be manually configured, and while it is fairly straightforward to aggregate updates from non-transit ASes, as these routes are essentially static stubs, this is not the case for routes from transit providers, where our attacker was assumed to be. In fact, a commonly used traffic engineering trick called *hole punching* assumes that transit providers do not forcibly aggregate each other's announcements. In hole punching, a router announces a path to both a prefix and a different path to a more specific prefix contained in the first. In this way the router can hint at different policies for this specific destination or can encourage load balancing. Using RouteViews data, we observed 569 core transit ASes actively using hole punching. The fact that hole punching is actively done is of great value to our adversary, as the way in which he builds prefixes makes them appear identical to hole punches.

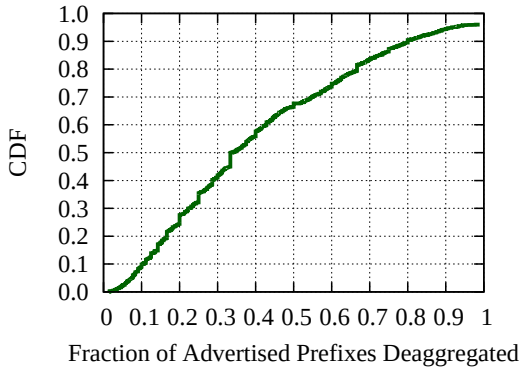


Figure 5.12: A CDF of the fraction of prefixes originated by transit ASes that could be aggregated into more general prefixes.

Moreover, one can examine RouteViews to see exactly how many ASes aggregate routes at all. By scanning RouteViews for ASes that advertise easily aggregatable blocks, for example `1.2.1.0/24` and `1.2.0.0/24`, we can quickly get a sense for how much aggregation is actually done *in practice*. We found that 100% of transit providers are observed advertising trivially aggregatable prefixes. Figure 5.12 shows a CDF of the fraction of advertised prefixes seen from transit providers that could be aggregated.

5.4.4 Prefix Count Limits

A different best practice that directly impacts our adversary is limiting the number of prefixes one router will accept from another. While there have been historical incidents that call into question whether a majority of ASes actually do this [49], let us assume the best case: that all ASes follow this practice. While some of the attacks covered in Section 5.1 center around sending a single malicious path to a target, others require the adversary to send sets of paths. Therefore, prefix limiting might present an issue for our attacker: if prefix limits prevent him from sending enough paths, his attack could fail. However, when examined more closely, this turns out not to be an issue. There are two different sets of prefix limits that will interact: those that the adversary's neighbors have set for it, and those that the victim has set with his neighbors. Somewhat counter-intuitively, the actual number of prefixes the attacker can push to a victim several hops away can be higher than the number of prefixes he can push to his neighbors. This is

because the attacker can set up multiple attack paths *that utilize the same first hop*. In this case the maximum number of malicious updates the attacker can send to the victim is the *sum* of the *victim's* prefix limits. We can examine our results from Figure 5.9c, where we examined the estimated number of attack flows an adversary would have to a given target, to get an estimation of the number of *per BGP session* advertisements an adversary would need to send for various attackers and targets. The results of this can be seen in Figure 5.13. As can be seen, in the worst case of a random attacker and target for more than half the cases, the adversary needs to send less than half of the current global routing table, something an adversary described in our threat model should be able to do. In the case of a tier 1 attacker against a tier 1 target, on average only 5000 prefixes per flow are sufficient.

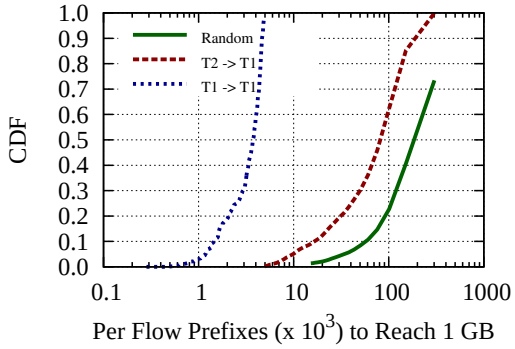


Figure 5.13: A CDF of the number of malicious updates required *per attack flow* to reach 1 GB of memory consumption for different classes of attacker and target nodes.

Another reason prefix limits do not have a large impact on the attacker is how large these prefix limits might be. The value of prefix limits depends on where the victim sits within the Internet topology. In general the victim falls into one of two places: either on the fringe of the network or not. If the victim is on the fringe of the network, then he is expecting to receive full BGP tables from a single digit number of providers, in which case his prefix limits are set at full table size (on the order of hundreds of thousands of updates). If the victim is not on the fringe, he might be expecting smaller amounts of updates from each individual peer, ranging from tens of thousands of prefixes up to full tables. However, victims who are not on the fringe of the Internet also have an

increase in their number of peers of an order of magnitude or more [17] compared to their counterparts in the fringes. This means that, even if we assume the core victim has prefix limits on the order of tens of thousands of routes, his aggregate route acceptance will be equivalent to that of the fringe victim, since the core victim has more peers. Lastly, it is advised practice to keep a safety margins of as large as 25% on prefix limits, so as to not accidentally exceed them [50]. This means our adversary can allow normal operation to continue, while using that safety margin to advertise his malicious routes.

5.4.5 Path Length Filtering

Recommended best practice is to limit the maximum accepted AS path length. Again, recent historical incidents call into question whether this is actually done [2]. Even if routers set a small AS path length limit (the current recommendation is 100 or less), we can recall back to Figure 5.4 in Section 5.1.2 and see that the length of the path is not a dominating factor for memory consumption in CISCO routers. With a path length of 22, each update accepted takes up 0.811 KB of memory. With a full path of length 253, we only see a marginal improvement to 1.108 KB of memory per update accepted. In our memory consumption attack, we cause a much greater memory consumption by adding community attributes to the updates.

5.4.6 BGPSec

Another possible technique that could hamper our attack is the widespread deployment of a cryptographically secured version of BGP. BGPSec [13], the most well known of such protocols, is currently in the preliminary stages of being deployed. BGPSec secures the validity of BGP paths and the authority of ASes to originate BGP advertisements. In BGPSec, ASes which are allocated blocks of IP prefixes from a numbering authority are given cryptographically signed tokens called a Route Origination Authorization [51], or simply ROA, that assert their ability to originate BGP advertisements for those prefixes. These ASes can in turn sign an ROA delegating a subset of those prefixes to another AS. When an AS originate a route it includes both this ROA and a signature covering both the destination of the route and the AS path. ASes receiving a route only consider it valid after confirming the signatures. When the AS forwards the route to another AS,

it again covers the destination, AS path, and all other signatures with a signature of its own. In this way a chain of trust is built protecting routes. The obvious question is how does this affect our adversary.

For starters, our adversary can not originate routes using *arbitrary* IP prefixes, as he only possesses tokens for the prefixes he owns. However, previous work by Goldberg et al [52] has demonstrated that even in a system where all users have deployed BGPSec, malicious routers can still launch prefix hijacking attacks by taking advantage of the preference for routes from customers. Essentially, the adversary takes a route it has learned about from a provider, and advertises the route to its other providers (the victims), a violation of no-valley routing. The victims will prefer the route if it prefix advertised by the route is not in its customer cone, and in turn advertise the malicious route to all of its BGP neighbors. Our adversary can use this same method to generate update messages beyond what he would have ROAs for.

The structure of the ROAs presents a possible method to halt the de-aggregation of IP prefixes that the adversary owns. ROAs contain a field stating the level to which the owning AS can de-aggregate an IP block, for example an AS might receive an ROA for `1.2.0.0/16` but the ROA could allow for advertisements down to a `/20`. This de-aggregation mechanism allows for ASes to utilize traffic engineering techniques covered in Section 5.4.3. More importantly from a protocol design standpoint, this provides a large efficiency gain, as a collection of ROAs can be aggregated into a single ROA. In theory the ROA could be used to prevent de-aggregation by restricting this field, causing de-aggregated routes to be considered invalid as they lack a correct ROA. This is unlikely to happen in practice for two reasons. First, it restricts the traffic engineering options available to operators. Second, in order to correctly cover all IP blocks exactly more ROAs must be stored than in the case where de-aggregation is allowed. For example, if an AS owns `1.2.0.0/16` and wishes to sell `1.2.128.0/17` to a second AS, in order to correctly reflect this IP block ownership ROAs for `1.2.0.0/17` and `1.2.128.0/17` must be issued instead of a single ROA for `1.2.0.0/16` allowing for de-aggregation to `/17`s. In additional to greatly increasing the complexity of issuing ROAs, under normal operating conditions the memory cost of these additional ROAs would outweigh the memory savings of forced aggregation.

Another capability that BGPSec hampers is our adversary's ability to arbitrarily

alter fields in BGP advertisements. For example, our adversary can not directly alter the AS path of a route to ensure it is unique, nor taint the AS path in order to contain attack flows. At a glance it would appear that this would halt our attack directly, but this is not the case. For example, consider the task of ensuring AS paths are unique and padding to the correct length. If an adversary controls a pair of routers in two different ASes then path distinctness can still be achieved by constructing a path that contains a unique permutation of the two ASes. Since the adversary controls both routers the correct signatures can be generated for this route. While semantically this AS path makes little sense, it is considered valid by BGPSec, as it is presented with the correct set of signatures. Additionally, not all fields are covered in a BGP advertisement are covered by a BGPSec signature. For example, the community attribute field is *not* protected, meaning an adversary can still alter it at will.

The largest negative impact BGPSec has on our adversary comes from its ability to make the identity of the misbehaving routers obvious. Since updates must have an unbroken chain of valid signatures, the malicious router must attach its signature to each outgoing attack update. Additionally, the adversary can not forge the existence of other ASes along the path, as it lacks the correct cryptographic keys to sign for the non-existent AS.

Chapter 6

Routing Capable Adversaries

Many distributed systems closely interact with the Internet's routing infrastructure. One such system is Decoy routing [53, 54, 55], an approach to building an anti-censorship tool. Instead of the traditional end-to-end based proxy solution, decoy routing instead places the proxies in the middle of paths, specifically at routers hidden throughout the Internet. Instead of explicitly connecting to these proxies, the user selects a destination whose path crosses a decoy router and signals to the router to act as a man-in-the-middle, proxying the connection to its real destination. This solves one of the main weaknesses of traditional proxies — enumeration and blocking by the censoring entity. Additionally, unlike traditional proxies, it is an explicit goal of decoy routing schemes to hide a client's usage of the system.

In this Chapter, we introduce the routing capable adversary, a new class of adversary against distributed systems which closely interact with the routing infrastructure. We will examine how routing capable adversaries function using the example of decoy routing. In this scenario, the routing capable adversary is a censoring authority who is capable of controlling how packets originating from its network are routed. We describe new attacks that can be launched by a routing capable adversary, and allow the censoring authority to defeat each of the security goals of decoy routing schemes. In particular, we show that a censoring authority, or *warden*, that has this capability can detect the network locations of decoy routers; we demonstrate that a warden in control of how a user's packets are routed can prevent those packets from being seen by the decoy routing system; we show how an adversary that can predict the properties of paths to innocent destinations can detect the use of decoy routing through timing analysis; and we show how that same warden can launch confirmation attacks that, by exploiting the differences between a normal user and a decoy routing user, test if a host is utilizing a decoy routing system.

The majority of the attacks we present focus on wardens who are able to exert control on how a user's packets are routed. In particular, to launch our attacks the warden must be able to locate decoy routers and select from a diverse set of paths in reaction to this knowledge. We show that a restrictive nation-state, an entity decoy routing was explicitly intended to defeat, presents exactly such an adversary.

Armed with both the knowledge of where decoy routers are located and a diverse collection of paths through the Internet, a warden is able to attack both the availability

and deniability of existing decoy routing schemes. In Section 6.3 we show how previous proposals for where to locate decoy routers allow a warden to find paths around them, preventing user traffic from being proxied. Worse, the warden can take advantage of the fact that while traditional hosts are not sensitive to the paths their packets take (a direct extension of the end-to-end principle), decoy routing users *are*. We will show a variety of ways a warden can detect this difference using active and passive means.

Finally, we show that there are fundamental difficulties based on the physical and economic architecture of the current Internet that limit the potential countermeasures to our attacks. We show that a deployment capable of denying these capabilities to a warden may be infeasible, requiring large fractions of the Internet to deploy decoy routers. Likewise, we discuss the limitations of traffic-shaping or other techniques in defeating timing analysis based on path properties. These limitations imply that while decoy routing may require a change in the tactics of censoring authorities, it is not an effective solution to the censorship circumvention arms race.

6.1 Decoy Routing Background

Internet censorship circumvention tools aim to provide users with unrestricted connectivity to network resources, even when those users are located in networks controlled by the censor, henceforth referred to as the *warden*. The mostly widely deployed censorship resistance tools used today combine proxies and encrypted tunnels, examples of which include Tor [56], JAP [57], and Ultrasurf [58]. These systems provide an end-to-end approach to circumventing Internet censorship. The user makes a connection to one of these services and the service then acts as a proxy, relaying traffic between the user and the censored destination.

Unfortunately, censorship authorities have responded to these schemes with increasingly sophisticated mechanisms for identifying the hosts providing this service; for instance, there is documented evidence that both China and Iran have at times applied sophisticated Deep Packet Inspection (DPI) techniques and, in the case of China, active network probing, to every outgoing TLS connection in an effort to identify Tor Bridges [59, 60]. Once these hosts have been enumerated, these systems are easily defeated by blocking all connections to their IP addresses. To solve this issue, *decoy routing*

systems were proposed. Decoy routing aims to fundamentally alter the way users communicate with the censorship resistance system.

Decoy routing systems [53, 54, 55], proposed concurrently by Karlin et al., Wustrow et al., and Houmansadr et al., use an end-to-middle approach to communication in an attempt to avoid being easily blocked. Instead of the censorship circumvention system being one of the endpoints in the communication, it is located amongst the routers used to forward packets on the Internet. Rather than making a direct connection to the proxy, the user instead selects an uncensored destination, called the *overt destination*, and initiates a TLS [61] connection to that host. The overt destination is selected such that the path from the user to the overt destination passes over a router participating in the decoy routing system, called a *decoy router*. The user signals the decoy router in a manner that the warden cannot observe, and the decoy router proceeds to act as a proxy, sending traffic not to the overt destination, but to the user's actual destination, called the *covert destination*. To the warden, it appears that the user has a functional TLS connection with the overt destination, when the user actually has a connection with the covert destination.

The details of how this is done vary based on the exact system being used. Currently, two implementations of decoy routing exist: Telex [54] and Cirripede [55]. In both systems, users signal their intention to use decoy routing by selecting random fields in packets (the TLS nonce in the case of Telex and the initial sequence number in the case of Cirripede), in a predictable, but unobservable, manner. The clients then proceed to complete a TLS handshake with the overt destination, while the decoy router acts as a man-in-the-middle, eventually extracting the negotiated cryptographic key. At this point the decoy router switches to proxy mode for this connection, terminating the connection from the perspective of the overt destination with a TCP reset, and extracting the user's covert destination from packets sent by the user. For more details on how these systems function, we refer the reader to the original works.

6.2 Routing Capable Adversaries

The goal of any warden is to prevent users from accessing a set of "forbidden" websites. This could be accomplished through a variety of means, such as dropping inbound or

outbound traffic, resetting TCP connections, or hijacking and middleboxing encrypted connections. A warden willing to make *routing decisions* in response to decoy routing systems can be considered a *routing capable adversary* (or simply a *routing adversary*).

Since an AS can simply change its policy configuration to alter which route it uses, and thus which path packets take, it is interesting to consider what tools this gives a warden. In addition to analyzing all traffic entering and leaving the network, a routing capable adversary is free to violate best practices and many assumptions about routing policy (e.g., those based on economic incentives, such as valley-free routing). As covered in Section 2.3, since routers store all currently valid routes, they can easily select between any of them for use in the forwarding table. Additionally, the warden could be selective about how it advertises routes to the rest of the Internet, to influence how traffic enters its network.

6.2.1 Wardens as Routing Adversaries

Since decoy routing was designed to defend against wardens as powerful as a nation-state, let us consider a variety of countries that have a history of monitoring Internet usage and censoring Internet access: Australia, China, France, Iran, Syria, and Venezuela. These countries also vary widely in the size and complexity of their network and their connectivity to the rest of the Internet.

Since a country can hold large amounts of political and economic control over the ASes operating within their borders, we can consider each to be not several individual ASes, but instead coalitions of ASes. While individual ASes within a warden country might have low degree in the Internet topology, collectively their connectivity to the rest of the Internet can be much higher. Using data from CAIDA [62] and the Berkman Center [63], we determined the size and connectedness of each country, as shown in Table 6.1. As an example, consider China with direct connections to 161 external ASes. This high degree of connectivity to the rest of the Internet means that China can select from up to 161 different paths *to any given destination on the Internet.* While other nations, for example Iran and Syria, are less well-connected, they still maintain a sufficient level of path diversity to perform routing attacks, as we will show in Section 6.3.

A wide variety of network engineering techniques can be used internally to allow a warden to take advantage of their path diversity. A warden could, for example, request

Country	ASNs	IP Addresses	PoC	External ASes
Australia	642	38,026,901	7	470
China	177	240,558,105	3	161
France	434	31,974,177	7	553
Iran	96	4,073,728	1	58
Syria	3	665,600	1	7
Venezuela	30	4,135,168	4	22

Table 6.1: The number of autonomous and IP addresses in each country, as well as the number of points of control (the smallest number of ASes that control 90% of IP addresses), and the number of external ASes directly connected to each country.

that an ISP black-hole traffic (advertise a route that is highly preferable to existing ones) to a target destination so that they can forward it out one of their external connections. Another possible mechanism would be to have all ISPs share MPLS VPN tunnels [64], allowing them to tunnel traffic for particular destinations to the desired external connections. No matter the exact mechanism, a warden has access to a potentially large number of unique paths for the majority of destinations, allowing it to act as a powerful routing adversary.

6.3 Routing Attacks

Decoy routing schemes have viewed the problem of selecting where to deploy decoy routers as an issue of *availability*. It is obvious that if a user does not have even a single destination whose path crosses a decoy router, he can not utilize the system. Moreover, a user needs to be able to locate such a path quickly. Overcoming these two challenges are where authors have focused in the past. The flaw in prior work is that it approaches these issues assuming that the warden is not an active adversary. However, as discussed in Section 6.2, wardens are not passive entities. In this section, we show how a warden can identify which ASes are running decoy routers, even in extremely large deployments. We then show how a warden is able to launch both active attacks against the availability of decoy routers and attacks that confirm if a user is utilizing a decoy routing system, defeating both specific security goals of these systems.

6.3.1 Detecting Decoy Routers

Some of our attacks require that the warden knows where decoy routers are deployed. In Telex [54], it is assumed that the directory of decoy routers is made publicly available, allowing clients to choose their overt destinations such that the usual path taken will cross a decoy router. While a public directory of decoy routers makes the use of decoy routing much simpler from the client's perspective, it also tells the warden which ASes are participating. Cirripede [55], however, instead relies on clients probing various destinations until they discover a path that crosses a decoy router. But even without such a public directory, the warden can still uncover which ASes are participating using an intersection-based discovery attack.

To determine which ASes are running decoy routers, the warden can probe a large number of paths to various destinations on the Internet using its own client. If the client does not connect to the decoy routing system using a path, the warden can add all ASes on that path to its list of "clean" ASes—the ASes that it knows are not running decoy routers. Using this list, the warden can proceed to look at all paths on which the client *was* able to connect. For each such path, the warden prunes out the known clean ASes, leaving only ASes which might be running decoy routers. If there is only a single AS left on such a path after pruning, then the warden knows that that AS must be running decoy routing (we refer to such ASes as being "tainted").

If more than one AS remains on a path after pruning, there are two possibilities. First, the warden can attempt to construct a new path for each AS remaining that otherwise only contains known clean ASes. As before, if the client fails to connect on these new paths, then that AS is also clean. If the client does connect, then that AS is tainted.

The second possibility is that the warden is unable to construct a new path. Note that the warden can always determine if the first AS on the pruned path is running decoy routing: they simply have the client attempt to connect to a destination inside that AS. From the perspective of the warden, this means that the later ASes on the pruned candidate path are "shadowed" by the first AS—any attempt to reach them goes through a tainted AS. To the warden, it then does not matter if they are clean or tainted.

To evaluate this and other attacks, we utilized the Nightwing Simulator, which was described in Section 3.3. The AS level topology utilized was taken from 2011. We ran our

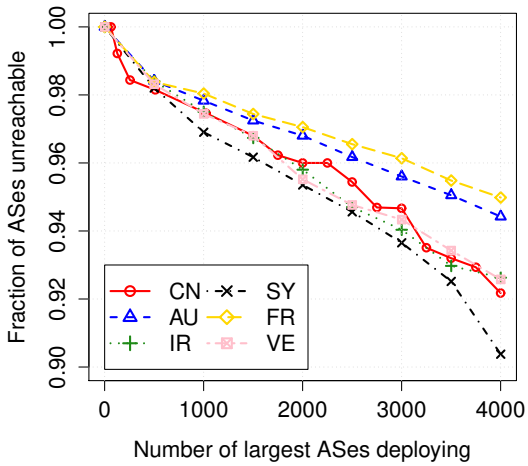


Figure 6.1: Fraction of ASes deploying decoy routers (chosen at random for various deployment sizes) that a warden can detect.

experiments for Australia, China, France, Iran, Syria, and Venezuela, considering each as a warden consisting of a coalition of all their member ASes, as covered in Section 6.2. After the routing topology converged, we then deployed decoy routers randomly to ASes for various deployment sizes, and measured what fraction of participating ASes each warden could detect using the method explained above. We found that all wardens had roughly equal success across all deployment sizes, and that they were able to detect over 90% of participating ASes for deployments as large as 4000 ASes, as shown in Figure 6.1. At such large random deployments, it is likely that most of the undetectable decoy routers were simply in the shadow of another decoy.

Since the warden must effectively mark all shadowed ASes as tainted, one goal of a decoy routing deployment would be to maximize the shadow produced by all participating ASes. However, as we explore in the following section, this is more difficult than it might appear.

6.3.2 Routing Around the Routers

As stated previously, the goal of decoy router deployment is to pick ASes such that all hosts in the warden's jurisdiction have at least one path that crosses a decoy router. Of

all previous work, Cirripede covers how to select ASes for deployment of decoy routers in the most detail. Houmansadr et al. cover two deployment scenarios: *random* and *Tier-1*. In the random scenario, they claim that only a small fraction of randomly chosen ASes, roughly 0.4% to 1.0% of all ASes according to their results, need to be selected. Alternatively, in the Tier-1 scenario, they claim that as few as two or three Tier-1 ASes are needed, since these large transit ASes will have a vast number of paths that travel through them, including many to popular destinations, making these paths easy to locate and use.

The problem with these evaluations is that wardens, especially large ones such as China, have a large collection of diverse paths for the majority of destinations. This means that when decoy routers are deployed to a handful of large ASes, all a warden needs to do is select paths to destinations that do not utilize these ASes. Essentially, routing capable adversaries redefine the concept of availability for decoy routers. Instead of needing a *single* path to a destination with a decoy router on it, *all* paths to a destination need decoy routers deployed along them. The reason for this is simple. If the warden has a collection of paths to a destination (some with decoy routers and some without), then all the warden needs to do is alter its routing policy to prefer routes that do not contain decoy routers.

Of course, if all paths to a destination have decoy routers then the warden is left with several options: refuse to send information to that network, launch detection attacks against hosts sending data to those networks, or middlebox a subset of TLS connections bound for those networks. China, the most interesting example of a warden, has shown a willingness in the past to cut itself off from parts of the Internet that take actions counter to their policies, but conceivably would be unwilling to apply one of those solutions to a large portion of the Internet. Egypt, during the Arab Spring of 2011, fully disconnected itself from the rest of the Internet temporarily, and Iran has recently raised the threat of building homegrown versions of popular websites and doing the same. In essence, the decoy routing availability problem boils down to finding sufficient ASes to deploy decoy routers such that it will be too costly for the warden to handle.

Using our simulator and our reconstructed Internet topology, we explored how large of a deployment is needed to successfully disconnect a warden from a large fraction of the Internet. We deployed decoy routers using a variety of deployment strategies and

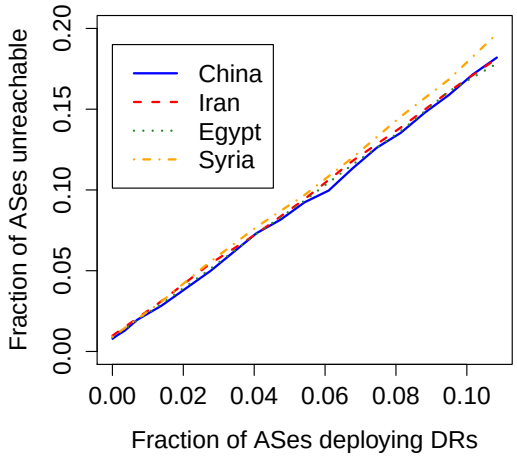


Figure 6.2: Fraction of all ASes unreachable for all wardens via at least one clean path when faced with deployments of decoy routers to random ASes.

measured the number of destinations to which each warden had at least one path that did not encounter a single decoy router, henceforth referred to as a *clean path*.

We start by considering Houmansadr et al.'s [55] "random ASes" scenario. Figure 6.2 shows the average fraction of destinations to which each warden fails to have a single clean path over 50 test deployments. This value represents the fraction of the Internet that each warden must cut itself off from in order to prevent use of the decoy routing system. We see that if deploying decoy routers to between 0.4% and 1.0% of all ASes, the wardens need only disconnect themselves from between 0.85% and 3.04% of the Internet. Essentially, these countries need only disconnect themselves from the ASes deploying decoy routers and an insignificantly sized "customer cone." Figure 6.2 also shows exactly what fraction of non-participating ASes (i.e. those that are not deploying decoy routers) are disconnected. As can be seen there, even if 10% of the Internet deploys decoy routers, they only disconnect the wardens from a mere 7-9% of the rest of the Internet on average.

We also consider the "Tier-1 only" deployment scenario. Figure 6.3 shows the fraction of the Internet that is unreachable as a result of deploying individually to each of the 100 largest (by degree) ASes, excluding the ASes in each warden that fall within that set. It is clear that this strategy fails to work, as in only 2.3% of all ASes are cut off from China in the best case, while the Egypt, Iran and Syria will be cut off from 9.7% on average. Figure 6.4 shows the fraction of destinations each warden is cut off from as a function of deploying *simultaneously* to the top N largest ASes. As can be seen, eventually this strategy will disconnect each warden from a large fraction of the Internet, but the deployment cost is quite high. For example, in order to cut China off from at least half the Internet all of the 96 largest ISPs in the world would need to deploy decoy routers to all exit points in their network, while still needing 74-78 of them to cut off much smaller countries such as Syria. We note that such a deployment would incur high equipment costs and require incentivizing a large number of profitable companies in diverse political settings.

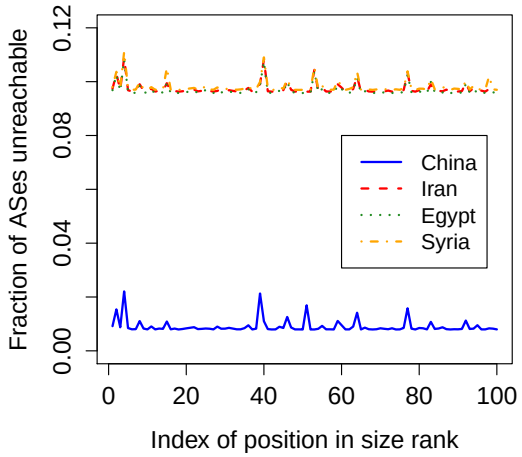


Figure 6.3: Fraction of all ASes which are unreachable via a clean path from wardens when decoy routers are deployed to each of the 100 largest ASes individually.

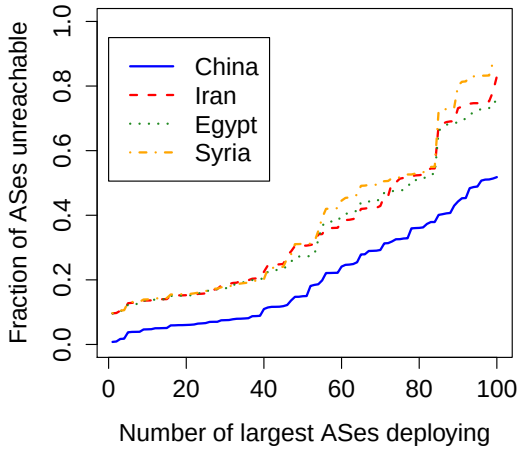


Figure 6.4: Fraction of all ASes unreachable via a clean path from the wardens when decoy routers are deployed to the N largest ASes collectively.

6.3.3 Detection Attacks

Attacking the availability of decoy routers is just one option open to the warden. Decoy routing systems also have the explicit goal of *unobservability*—hiding the fact that a host is using the system. However, wardens with path diversity are capable of launching attacks that unmask users of decoy routers. While the availability attack of Section 6.3.2 requires little in the way of real time actions by the warden (nothing more than a handful of lines in the configuration files of routers), the attacks of this section have a much more active element. In these attacks, the warden intentionally selects some paths to destinations that cross at least one decoy router, henceforth referred to as *tainted paths*. The warden then utilizes the state and topology of the network to identify a decoy routing user.

TCP Replay Attacks

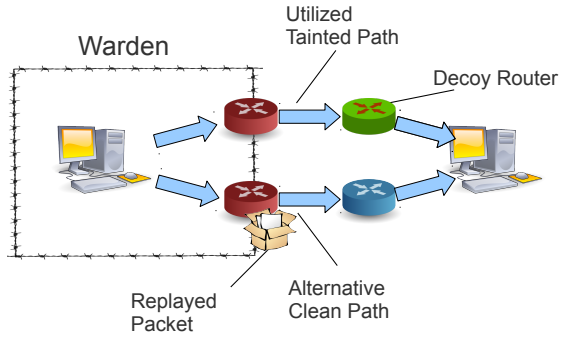
Consider two hosts sending packets to a destination, one utilizing decoy routing, ostensibly sending traffic to the overt destination, the other a host legitimately communicating with that same destination. The most obvious difference between these two hosts is that

the latter actually has a TCP connection with the destination while the former does not. The decoy routing user started a TCP connection with the overt destination, but in both existing decoy routing schemes that connection is torn down with assistance from the decoy router after TLS negotiation.

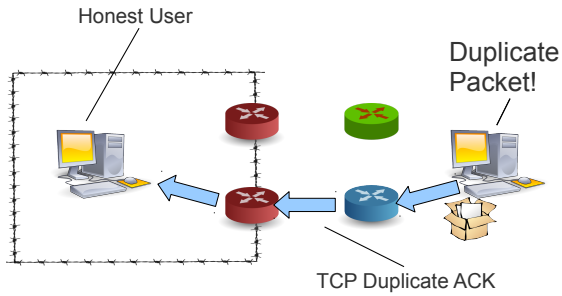
The challenge for the warden is to come up with a way to test if the destination thinks it actually has a TCP connection with the host. It turns out that the warden can do this quickly and cheaply if it also has a clean path to the destination, as shown in Figure 6.5. The warden need only replay a TCP packet sent by the host, but instead of forwarding it along the tainted path that the host is using, the warden forwards it along a clean path (Figure 6.5a). Because there are no decoy routers along the path to intercept the packet, it will reach the destination, and, by the end-to-end nature of the Internet, the destination is agnostic to the actual path taken by the packet. If the host was a legitimate host (Figure 6.5b), that is, not using decoy routing, then because there is an existing TCP stream, the destination will treat this packet as a duplicate, and, per the TCP RFC [65], send a duplicate acknowledgment. On the other hand, if the host was actually using decoy routing (Figure 6.5c) and the destination was simply the overt destination, no TCP connection will exist, and the destination will respond with a TCP reset packet.

We note that if the return path of the packet crosses a decoy router, that decoy router could drop the packet.[1] However, the warden has multiple ways to force asymmetry of inbound and outbound paths.

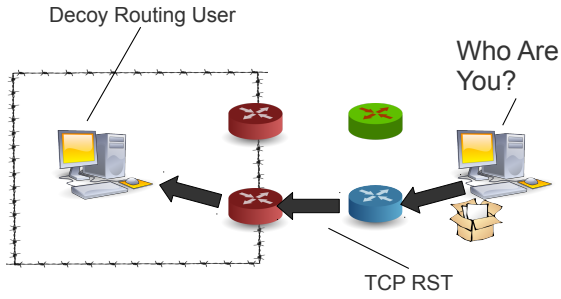
[1] In our understanding of the Cirripede design, the state of all client connections is replicated to all decoy routers, providing this functionality, while Telex does not currently explicitly provide this functionality.



(a)



(b)



(c)

Figure 6.5: Illustration of a simple confirmation attack launched using replayed TCP packets. In Figure 6.5a the warden has both a tainted path and clean path to a destination, and allows users to utilize the tainted path. The warden then replays an observed TCP packet using the clean path. If the user is honest (Figure 6.5b), a duplicate acknowledgment is seen. If the user is a decoy routing user (Figure 6.5c), a TCP reset is instead seen.

Forced Asymmetry

Asymmetry in the path taken by data going between two hosts on the Internet exists naturally [66]. However, a warden is able to artificially induce path asymmetry on a far larger scale. At the simplest level, all a warden needs to do is intuit which path a destination network is utilizing to send traffic to the warden, and then alter its routing policy to ensure that it picks a different path to the destination. The warden can utilize a variety of metrics including inferred AS relationships, incoming router/interface, TTLs, and packet timings in order to determine which route a destination is using.

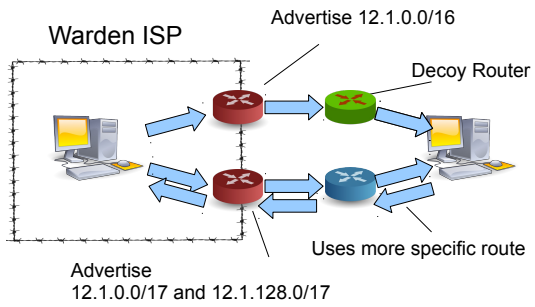


Figure 6.6: An illustration of how a routing adversary creates a clean asymmetric return path. The destination will, when presented with a more specific route, selects the longer prefix.

Alternatively, a more active warden can utilize BGP's loop avoidance mechanism [8] in order to force both return path asymmetry and ensure that the return path is free of decoy routers. This attack relies on a traffic engineering technique known as hole punching, as shown in Figure 6.6. In hole punching, a router advertises both a block of IP addresses and a de-aggregation of that block, each with different path properties. Since these IP blocks are technically different, BGP will treat them as routes to different destinations, allowing for more specific policies for certain blocks of IP addresses. These more specific routes will automatically be used, as routers always forward on the *most specific matching IP block*. The warden then, for every block it wishes to advertise, hole punches a second set of routes covering the entirety of each block it would normally advertise. Since there is no currently deployed mechanism to prevent a router from

falsifying route properties, an active warden can add every known decoy router deploying AS to these more specific routes. When a decoy router deploying AS receives these routes they will drop them, as it would appear like they would be creating a loop, but ASes which do not deploy decoy routers would not find themselves in the path already, and so would accept and forward these routes as normal. Since these routes are more specific, even if these non-decoy routing ASes also have the more general route that travels through decoy routing ASes, it will instead select the more specific clean route.

No matter how the warden achieves path asymmetry, the results are damaging to decoy routing systems. In the case of Telex, the decoy routing system simply ceases to function, as it requires path symmetry. Cirripede would function, but its use would become obvious. As shown in Figure 6.7, packets returning from the decoy router will enter the warden at a different location in the network compared to packets returning from the overt destination. If all return paths are tainted, a decoy routing system could, in theory, shuffle packets between decoy routers to cause them to enter at the correct router and interface with the correct TTL, but this would further simplify timing attacks.

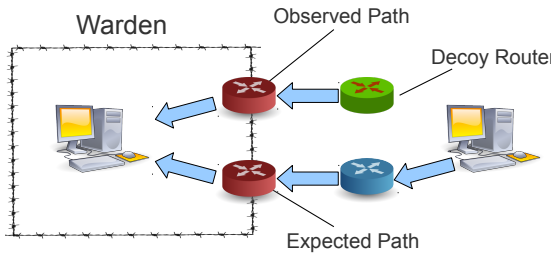


Figure 6.7: An illustration of why path asymmetry makes decoy router usage obvious to the warden. Since traffic must be sent from the decoy router to the user, if the path from the decoy router to the user and from the overt destination are disjoint, then the warden will see packets arrive on a different interface then what is expected.

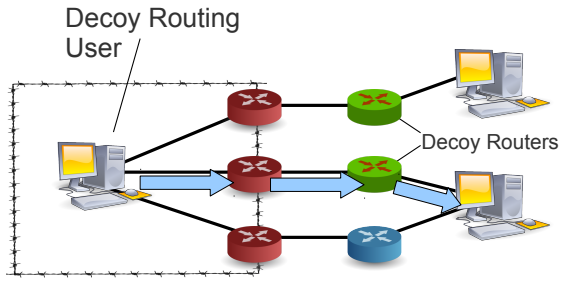
The "Crazy Ivan" Attack

Another active attack for confirming if a user is utilizing a decoy routing system we call the "Crazy Ivan" Attack. A Crazy Ivan was a maneuver utilized by Soviet submarine

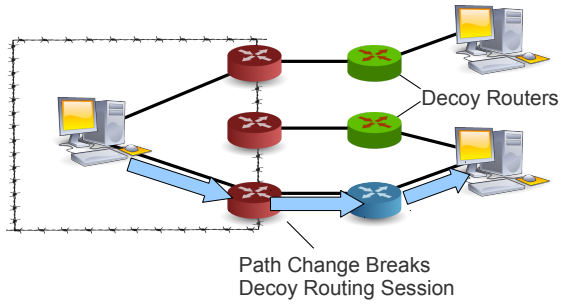
commanders during the cold war. It consisted of a series of radical course changes in an effort to determine if an enemy submarine was hiding behind his submarine, in an area that is acoustically masked by engine noises, called a submarine's baffles. In an analogous manner, the warden can initiate a series of radical path changes and withdrawals in an attempt to unmask decoy routing users.

Again consider both a user who is utilizing decoy routing and a user who is not. Both are currently sending traffic down a tainted path. Now consider what happens if the warden flips the path utilized to this destination to a clean path. Any host not using decoy routing will not be impacted by this, and will continue with his session. Decoy routing users, however, will be impacted, as their functionality is sensitive to the path. In the worst case for the user, behavior similar to that discussed in Section 6.3.3 is seen—TCP reset packets sent from the destination. Even if the return path crosses decoy routers, which can drop the reset packets, the decoy routing user is presented with an issue. His decoy routing session no longer functions, and he can no longer pretend to communicate with the overt destination. While observed user behavior after the path to the destination is no longer tainted is not definitive proof of decoy router usage, this experiment can be repeated multiple times until the warden has a high enough confidence in its conclusions. A graphical representation of this attack can be seen in Figure 6.8.

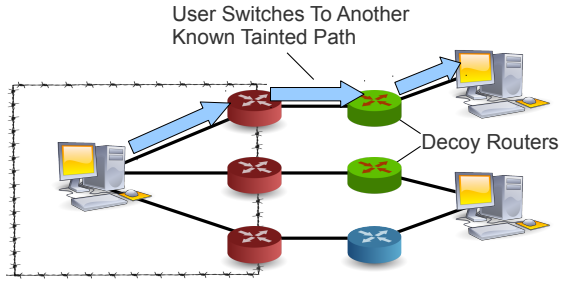
Of course there is the question of what an adversary does when no clean paths are available. First, it is clear that destinations to which an alternate clean path can not be found are sub-optimal honey pots. If the warden is pushed into a scenario where such routes must be utilized another option still exists. The warden could, instead of changing the path to a destination, stop forwarding packets to the tainted destination all together. This will obviously disrupt both honest hosts and decoy routing users. The difference is that honest hosts will start new sessions with random destinations, while the decoy routing user will attempt to start new sessions down tainted paths. Again, repeated iterations of this experiment can be done to test if a user is utilizing decoy routing. Investigating the effectiveness of this last attack involves modeling user behavior and browsing habits, making it outside the scope of this work.



(a)



(b)



(c)

Figure 6.8: An illustration of the Crazy Ivan attack. In Figure 6.8a, the warden allows users to utilize a tainted path. In Figure 6.8b, the warden switches to a clean path, breaking decoy routing user's session while leaving honest users unaffected. In Figure 6.8c, the user begins a new session, using another known tainted path, implying the users is looking for a tainted path. The warden repeats this tests several times to establish confidence in this assertion.

Chapter 7

Economic Routing Attacks

The Decoy Routing systems discussed in Chapter 6 are not the only systems that manipulate higher layers of network functionality from positions in the middle of the network. We will use the collective term *Traffic Manipulating Boxes*, or TMBs, to describe such systems. While some TMBs exist to provide simple improvements to the transit functionality of the network, others such as the previously covered Decoy Routing, traffic shaping [67, 68], and active surveillance boxes such as the NSA's alleged QUANTUM-INSERT project [69, 70, 71] seek to use privileged position in the Internet to observe and manipulate traffic in a manner not intended by either its sender or the AS that hosts its sender.

As shown in Chapter 6, large ASes and nation-states, what we have termed Routing Capable Adversaries, can use their capability to select which paths they utilize, and which they advertise to others, to launch a straightforward attack on the traffic coverage of TMBs by simply routing their traffic around deployers – ASes that have elected to deploy those boxes – and refusing to send traffic to any destination reachable only through deployers. This attack, termed the RAD (Routing Around Decoys) attack, defeats TMBs since they can neither observe nor manipulate traffic that does not cross them. However, Houmansadr et al. [72] pointed out that ASes launching a RAD attack incur added costs, possibly making the RAD attack prohibitively expensive.

In this Chapter, we consider how economic forces can work *for* the resistor, rather than against it. Instead of attempting to directly attack the availability of TMBs, our resistors inflict economic damage on deployers, while at the same time incentivizing ASes which elect to not become deployers. Assuming transit ASes are rational economic entities, these incentives will cause deployers to remove TMBs, accomplishing the resistor's goal of minimizing exposure to TMBs. This alternative approach to attacking availability has the added benefit of not requiring the resistors to disconnect themselves from portions of the Internet. In order for this strategy to be viable we must accomplish two distinct goals.

First, we must adjust how the RAD attack from Chapter 6 functions in an effort to remove the costs to the resistor. The original RAD attack had no concern for the resistor's costs; namely, added infrastructure costs from turning customer ASes into transit ASes, using paths learned from providers over those from customers, and increased path lengths. We present three alterations to the RAD attack which adjust how the attack

is executed to successively eliminate each of these costs. This reduction in resistor costs is not achieved for free, as each of these adjustments reduces the ability of the resistor to find paths free of TMBs. We evaluate these altered RAD attacks, along with the original RAD attack launched from four different resistors, against several deployments, including the deployment proposed to defeat the RAD attack. We find that our lower cost RAD attacks are still able to find paths free of TMBs, and therefore able to migrate a non-trivial amount of traffic away from deployer ASes.

Secondly, we need to maximize the economic costs the resistor can inflict *on the deployers*. Since TMBs require a privileged position on the Internet, deployers need to be large transit ISPs. These ISPs generate revenue by carrying network traffic, the exact thing a routing capable adversary can manipulate. We propose a new form of RAD attack with the goal of coercing deployers to remove TMBs by inflicting economic loss. This attack differs from the original RAD attack, in that it does not *directly* reduce the exposure to the TMBs themselves. This style of attack has a desirable property: the resistor does not have to cut itself off from any destinations on the Internet, it can instead tolerate TMBs functioning while economic pressure builds. In order to evaluate this, we develop a model of Internet scale traffic flow, and build an approximation between our simulated traffic and real world ISP revenue. Our results reveal that routing capable adversaries are extremely powerful economic entities, the largest of which are capable of inflicting a billion dollars in lost annual revenue simply by making different routing decisions.

7.1 Definitions

Traffic manipulating boxes, or *TMBs*, are middle boxes deployed to Autonomous Systems (ASes) that make up the transit core of the Internet. These devices utilize privileges in the transit core to observe or manipulate traffic *not* originated by the owner of the middle box. As covered in the introduction, there are several examples, both proposed and deployed, of TMBs, including decoy routing [53], surveillance devices which actively manipulate traffic in flight [69, 70, 71], and traffic shaping boxes [67, 68].

We refer to the organization that actually wants the TMBs placed on the Internet as the *originator*. The originator places TMBs inside the infrastructure of willing or compelled ISPs, which we refer to as *deployers*. It is of course possible for the originator and deployer to be the same organization; however, this is often not the case. For example, in the case of decoy routing, an outside organization concerned with providing censorship circumvention would be the originator, not the ISPs hosting the Decoy Routers.

Originators have several costs, for example the price of hardware, which they have to pay on a per deployer basis. This means that they have to attempt to minimize the number of deployers while still maximizing the amount of traffic that the TMBs can manipulate. In order to do this, the originator needs to select deployers that will see not only traffic addressed to themselves, but other ASes as well, hence the requirement that TMBs be placed in the transit core of the Internet. Additionally, originators will also have to defray economic losses that deployers suffer as a result of installing TMBs. These side payments would resemble those allegedly provided by the NSA to RSA Labs for the deployment of specific technologies [73]. These side payments will be covered in detail in Section 7.4.

7.2 Methodology, Resistors, & Deployment Models

7.2.1 Routing Model

To evaluate both the routing capabilities of new resistor strategies and the economic impacts of the resistor actions, we again utilized the Nightwing simulator covered in Section 3.3. The topology used in the simulation is from Caida's inferred AS relationships dataset taken from February of 2015 [62]. This topology contains 49,755 unique autonomous systems.

7.2.2 Traffic Model

In order to measure economic losses, we simulate traffic flow through the Internet using the pre and post RAD attack paths. To achieve this, we needed an Internet scale model of where traffic originates from, where its destination is, and how much of it there is. We based our model on existing work, specifically that of Gill *et al.* [74], supported by

the measurements of Labovitz *et al.* [75], the World Bank [76], PeeringDB [77], and Sandvine [78]. This model takes into account both the random host to host traffic distributed across Internet hosts and traffic which is concentrated at extremely large content providers such as Netflix and Google, so called "super ASes."

Simulated traffic units fall into one of two different buckets. Traffic in the first bucket is marked as "host to host" traffic, meaning that it neither originates from, nor is addressed to, large scale content providers. The second bucket of traffic represents traffic that originates from large scale content providers and CDNs. Measurement studies have shown that the overwhelming bulk of this traffic flows from the CDNs to end hosts. In order to build a model of relative traffic ratios between these two types of we consulted the Sandvine Global Internet Phenomena Report [78], which provides a region by region breakdown of an average user's Internet traffic. The percentage of traffic from large scale content providers ranged from 30.0% percent for users in the African region, up to 67.5% percent for users in the North American region. The breakdown of the most popular sources of content by region were provided by Sandvine as well. These sources generally included Youtube/Google, Netflix and Facebook. Traffic not attributed to a specific destination was spread evenly between several other large CDN providers noted by previous measurement studies: Microsoft Azure, Akamai, Limelight, and Baidu. In order to model which ASes have CDN nodes locally, we referenced Netflix's public documentation for how it selects ASes to host content providing servers [79].

In order to establish the relative values of traffic leaving and entering ASes three data sets were combined. Sandvine provides the amount of bandwidth consumption from an "average" user in various regions. This information was combine with the World Bank's estimation of the number of Internet users in each country to get relative inbound and outbound bandwidth on a per nation state basis. In order to assign that bandwidth to ASes, we first assigned each AS to the nation state it primarily resides in. We then consulted PeeringDB, which is a system that allows ASes to advertise their willingness to peer with other ASes. ASes which elect to participate in PeeringDB have the ability to optionally disclose the average amount of inbound and outbound bandwidth from their AS that peers should expect. Of the roughly 49,000 ASes which exist in our topology just over 6,000 report bandwidth estimates. In order to establish relative bandwidth values between all ASes a Random Subspace classifier was trained based on

Resistor	Member ASes	Adjacent ASes	Provider ASes
Brazil	3140	162	57
China	255	292	59
Germany	1792	1372	142
Iran	399	21	14

Table 7.1: A comparison of relevant statistics for example resistors featured in simulations.

AS features including AS degree, size of customer cone, primary country of operation and size of IP space advertised. The resulting classifier had a correlation coefficient of 0.71. Nation state bandwidth was divided proportionally between member ASes based on this estimate of inbound/outbound bandwidth to peers.

Our experiments focus on *billable traffic*, rather than total traffic. Billable traffic is every unit of traffic an AS either receives or sends to a customer AS. We focus on this traffic since each unit of billable traffic directly corresponds to *revenue* generated by the transit AS, a result of the transit AS being paid by the associated customer for carrying the traffic. The amount of money a customer pays to have that traffic delivered is typically established based on peak bandwidth consumption. Intuitively, peak load should be proportional to the total traffic sent in our simulation, thus total costs will be proportional to simulated billable traffic units in our experiments. The telecom industry consulting group TeleGeography reported that total annual IP transit revenue in 2014 was 4.9 billion US dollars [80], allowing us to compute a ratio between simulated traffic units and annual revenue of 1.66652×10^{-20} USD of annual revenue per simulated traffic unit.

7.2.3 Resistors

We have selected four different resistors as examples. A brief summary of their properties can be seen in Table 7.1. Brazil is the largest in terms of member ASes, but the second smallest in terms of unique ASes it connects to. Our selection of Brazil was motivated by recent fallout over the NSA QUANTUM program. It is not far fetched, given current proposals [81], that Brazil would elect to act as a resistor with relation to surveillance related TMBs. China has a highly developed Internet infrastructure and has demonstrated a willingness to take radical steps to attack censorship circumvention systems

in the past. Germany has expressed responses similar to Brazil's when it comes to US government surveillance. However, Germany enjoys an order of magnitude more unique points of connectivity with the rest of the world, and 24 times the number of unique providers compared to Brazil. Iran represents the least well connected, and consequently weakest, of our resistors.

7.2.4 Deployment Selection

In this work we focus on two types of deployments. First, we consider a *targeted deployment*, where the placement of TMBs is done with the goal of maximizing the amount of traffic from the resistor which travels across tainted paths. Censorship circumvention systems are examples of TMBs that might be deployed in such a manner. For example, one could envision a theoretical deployment of decoy routers targeting a repressive nation state such as China. Targeted surveillance is another form of TMB that might utilize this deployment strategy. In order to select which ASes to deploy to, our originator first infers what BGP paths the resistor is utilizing and weights those paths based on an estimation (or measurement) of the amount of traffic utilizing that path. Potential deployers are then scored based on the weights of the routes they appear on. The candidate AS with the largest score is then selected as a deployer, all routes which are tainted by that deployer are removed from the set of resistor routes, and the other candidate deployers are re-scored against the resistor routes which remain clean. This process is repeated until the desired number of deployers has been selected. It is worth noting that in both this deployment and the following deployment we did not limit our originator to selecting only ASes that reside in one nation, instead we allowed the originator to deploy to any AS outside of the resistor. For all of our resistors, a deployment of 10 TMBs resulted in more than 60% of the traffic leaving the resistor initially utilizing a tainted path. When deployments were larger than 40 TMBs all resistors saw 80% of their traffic traveling through at least one TMB. Figure 7.1 shows the percentage of traffic leaving our resistors that utilizes a tainted path initially when targeted deployments are constructed in this manner for each resistor.

We also consider a *global deployment* where the originator has the goal of maximizing the amount of traffic passing over at least one TMB regardless of the traffic's source or destination. This type of deployment would be utilized by mass surveillance programs.

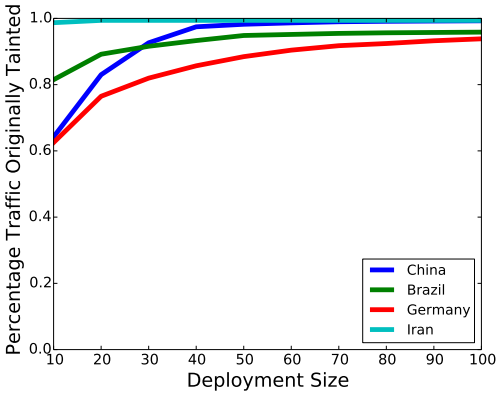


Figure 7.1: The fraction of traffic originating inside our resistors which crosses at least one TMB placed in a targeted deployment.

Selection of deployers for the global deployment occurs in the same manner as it does in the case of a targeted deployment with one change. Rather than only considering paths utilized by the resistor, all paths from all ASes are considered. This means that, unlike the targeted deployment where there is a different deployment for each resistor, there is only a single unique global deployment. This global deployment resulted in slightly less traffic from our resistors crossing at least one TMB, but still resulted in 80% of resistor traffic utilizing tainted paths when at least 40 TMBs are deployed. The percentage of traffic using tainted paths initially against our global deployment is shown in Figure 7.2.

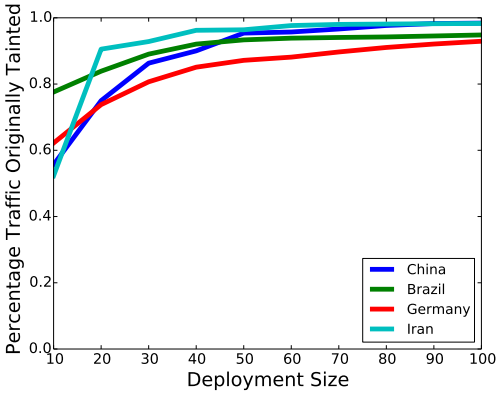


Figure 7.2: The fraction of traffic resistor originated traffic which crosses at least one TMB deployed as part of a global deployment.

7.3 Reducing Resistor Costs

It is natural to ask if there is some way to reduce the costs placed on resistors while not crippling their ability to avoid tainted paths. In this section, we examine how a resistor can launch less costly RAD attacks by altering where it considers the existence of TMBs in the BGP route selection process. We will evaluate the strength of these attacks by examining how much traffic they can move away from deployers compared to the original RAD attack.

7.3.1 Levels of Resistor Strength

The BGP route selection process allows for multiple locations where a resistor can insert logic that takes into account TMBs. In the original RAD attack TMBs were taken into account before local preference, essentially adding a rule of "Prefer clean path" to the decision process. Additionally, a rule was added to the advertisement decision process forcing an AS to advertise *all* best paths to an adjacent AS if that AS was also a resistor. Prior work [72] pointed out that resistors will suffer increased costs as a result of launching the original RAD attack. First, the change to route advertisement policy would force some non-transit ASes to act as transit providers, carrying traffic for other ASes. Second, by considering path cleanness before local preference, ASes might experience increased transit costs, as some destinations reachable via routes learned from customers would shift to non-tainted routes learned from providers. Next, since path cleanness is also considered before AS level path length, the resistor might select some routes that have longer AS level paths, possibly resulting in lower Quality of Service. Lastly, the resistor will be forced to increase the traffic load on certain links in the topology, possibly requiring the upgrading of the physical network infrastructure.

In this work we present three alternative resistor strategies which seek to reduce these resistor costs. Each of these strategies is successively less costly than the last, but also less able to find non-tainted paths. A summary of the reduced costs of each resistor strategy can be found in Table 7.2.

First, a **Local Preference Resistor** removes the additional advertisement logic, leaving route selection the same. By allowing ASes to use normal advertisement rules, non-transit ASes will never be forced to become transit ASes. However, the resistors

	Original	**Local Pref**	**Path Length**	**Tiebreak**
Transit Conversion	Yes	No	No	No
Increased Cost	Yes	Yes	No	No
Increase Path Length	Yes	Yes	Yes	No
Increased Link Load	Yes	Reduced	Reduced	Reduced

Table 7.2: A comparison of the costs incurred by a resistor using various strategies. The original Routing Around Decoys (RAD) attack and the proposed resistors we evaluated are shown.

lose the guarantee that if at least one resistor has a clean path all resistors will have a clean path.

A **Path Length Resistor** alters when clean paths are preferred over tainted ones. Instead of doing this before Local-Pref, we could do it after Local-Pref but before AS path length. This would mean that in addition to the cost reductions of the Local Preference Resistor, an AS would never find itself in the position of using routes from a provider in preference to routes from a customer, preventing the resistor from experiencing increased transit costs.

The **Tiebreak Resistor** takes this a step further moving the check for TMBs to the end of the route selection process, directly before tie-breaking. This would, in addition to the gains of the previous two resistor strategies, eliminate selecting longer AS level paths.

7.3.2 Evaluation of New Resistors

In order for the resistor to inflict economic harm on deployers, it must be able to successfully transition traffic from tainted paths to clean paths, a process we refer to as *deflecting traffic*. If our resistor is able to successfully deflect traffic away from deployers, the deployers will suffer economically. The originator of the TMBs will in turn have to defray the costs suffered by deployers. It should be noted that unlike the original RAD attack in Chapter 6, in this scenario the resistor does not need to find clean paths

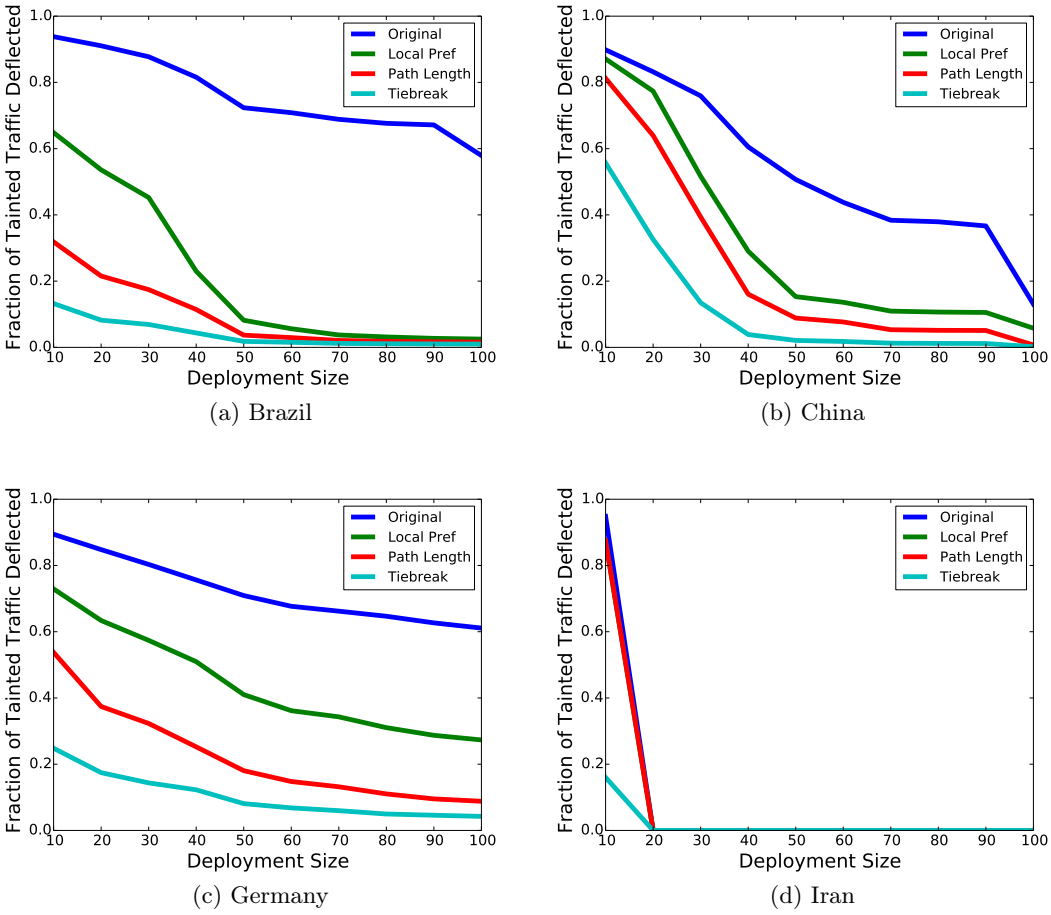
to *all* destinations. Instead, the resistor needs to replace a *sufficient* number of tainted paths with clean ones to create noticeable economic harm to the deployers in an effort to incentivize them to remove the TMBs. A discussion of how this new strategy works against Decoy Routers and other systems where the existence of a single tainted path is sufficient for the system to function is done in Section 7.5.

In this section, we cover how much tainted traffic our resistors were able to deflect away from deployers, and what the consequences were for path Quality of Service. Financial costs to the resistor will be analyzed with the costs to the originator of the TMBs in Section 7.4. We simulated all four of our sample resistors, utilizing each of our newly proposed RAD strategies and the original RAD strategy, against both targeted and global deployments of various sizes.

Ability to Deflect Traffic

The fraction of traffic which originally crossed deployers that was successfully deflected by each RAD attack can be seen in Figure 7.3 and Figure 7.4 for targeted and global deployments respectively. As can be quickly seen from the figures, the more costly RAD attacks are more capable of deflecting traffic away from deployers in all deployment scenarios. However, even the weakest of the new RAD attacks manages to deflect some amount of traffic in all scenarios except targeted deployment against Iran.

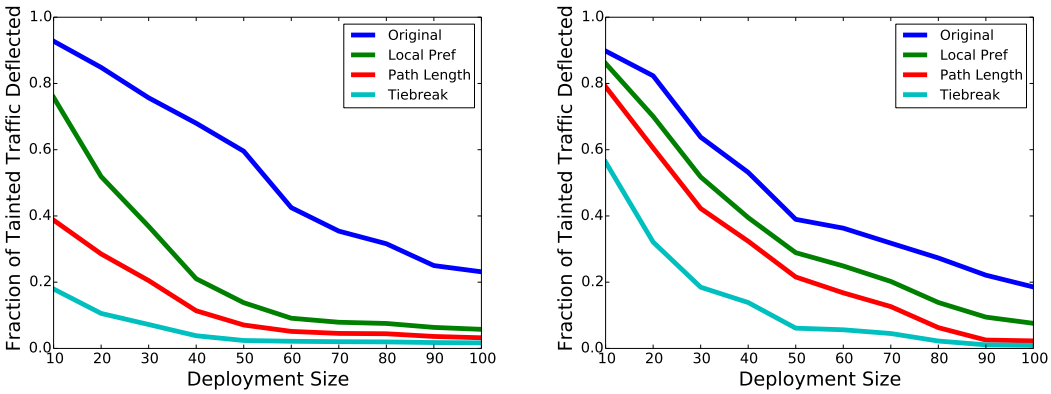
The more well connected resistors are able to deflect traffic to varying degrees against both deployment models using all of the RAD strategies outlined in Section 7.3.1. In the case of China and Brazil the Tiebreak strategy is quite weak, approaching zero deflected traffic, while Germany still manages to deflect between 5% and 22% of tainted traffic using this no cost strategy. The Path Length strategy, which is the strongest financially free strategy for the resistors, can deflect a large amount of traffic at smaller deployment sizes, but can still deflect, depending on the resistor, between 1% to 9% of tainted traffic for deployments of 100 ASes. Resistors gain between a 100% and 200% increase in deflected traffic on average by stepping up to the Local Preference strategy in exchange for monetary costs. The same level of gains are seen moving from the Local Preference strategy to the original RAD attack with the exception of Brazil, which massively benefits from using the original RAD attack, a result of the small Brazilian ASes multi-homing with ASes located outside Brazil.



(a) Brazil (b) China

(c) Germany (d) Iran

Figure 7.3: Measured capability of our three lower cost RAD strategies, along with the original RAD attack to deflect network traffic from tainted paths, resulting from deployments targeted toward each resistor, to clean paths. Note in Figure 7.3d that at deployment sizes of 20 or greater Iran is completely surrounded.



(a) Brazil (b) China

(c) Germany (d) Iran

Figure 7.4: Simulated ability for our collection or RAD strategies to deflect traffic in the face of a deployment of TMBs which aims to maximize the total traffic observed, regardless of source or destination.

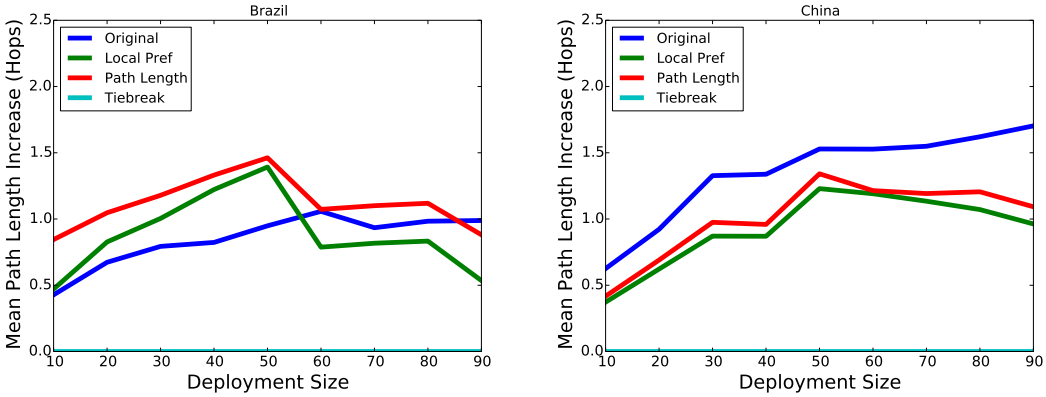
Iran is unable to deflect traffic against targeted deployments of size 20 or greater since they completely surround Iran. How Iran, or any other resistor which finds itself in this scenario, is able to inflict economic harm on the deployers will be covered in Section 7.4.3. When considering a global deployment, even with its small pool of available alternate routes, Iran is still able to deflect traffic away from deployers. For the other resistors, the global deployment is slightly more resilient to RAD attacks compared to targeted deployments. Iran also has the only instance of two of our RAD strategies being essentially equivalent. The Local Preference and Path Length strategies return nearly identical results against a global deployment, a result of limited path choices being available to the resistor.

Quality of Service

Three out of the four tested RAD attack strategies can result in increased AS level path lengths. AS level path length, while not a direct measure of latency or path reliability, has been utilized by prior work [72] in this area as a rough measure of path quality of service. For the sake of comparison we include measurements of mean path length increase for paths which were changed during the various RAD attacks against global deployments in Figure 7.5. Results are similar for targeted deployments and can be seen in Figure 7.6.

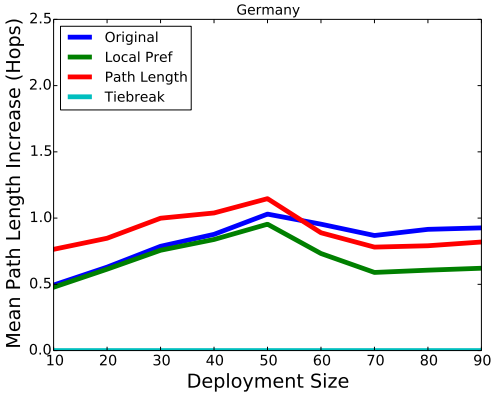
In all cases, Germany sees a mean path length increase of less than 1 AS. Brazil is the same with the exception of the original RAD attack against a targeted deployment, which has a mean increase of 1.5 ASes. China sees an increase in these same ranges for the Local Preference and Path Length strategies, but sees a mean increase of roughly 2 hops for the original RAD attack when deployment sizes are large. Iran sees no path length increases for targeted deployments as the resistor does not change paths when no clean paths can be found. Confirming that which was evident by definition, the Tiebreak strategy results in no path length changes in all cases.

More accurate measurements of path quality are difficult to achieve. We attempted to reproduce the latency experiments of Houmansadr et al. [72], but were unable to produce estimates for the latency of even 0.1% of the changed routes using the method and data set described in [72]. Accurately estimating the latency of unused BGP routes seems to be an interesting question that is beyond the scope of this work.

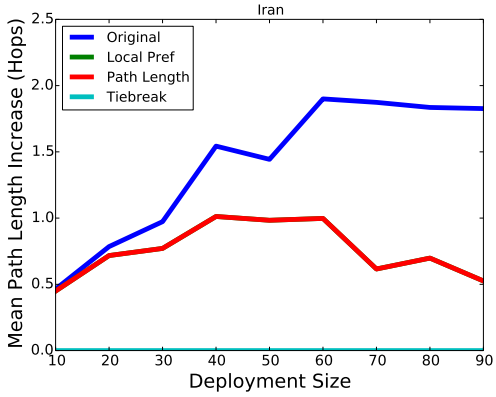


(a) Brazil

(b) China

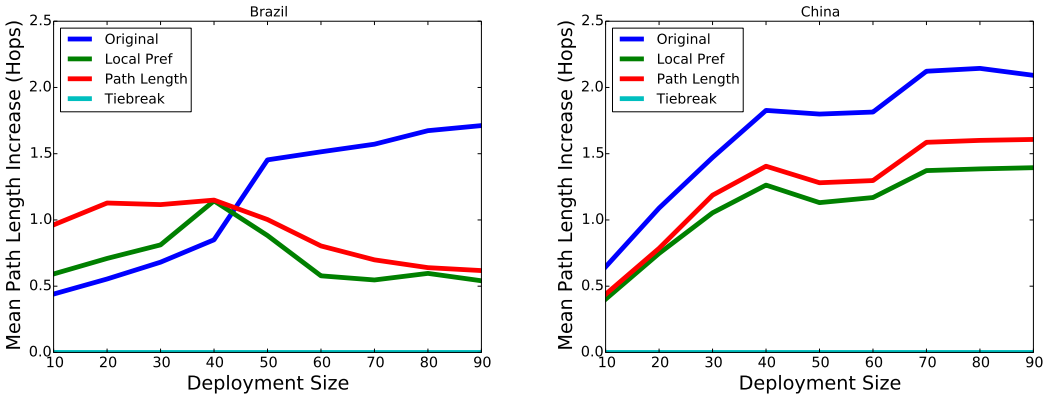


(c) Germany



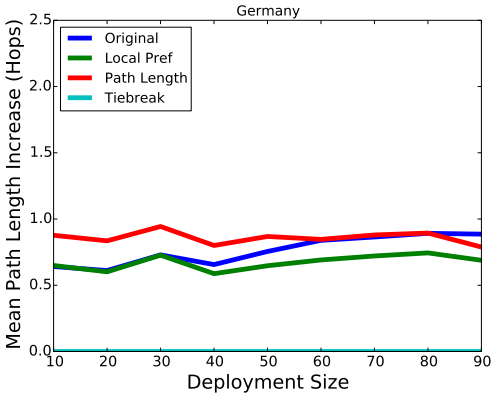
(d) Iran

Figure 7.5: Mean AS level path length increase see attempting to route around a global deployment.

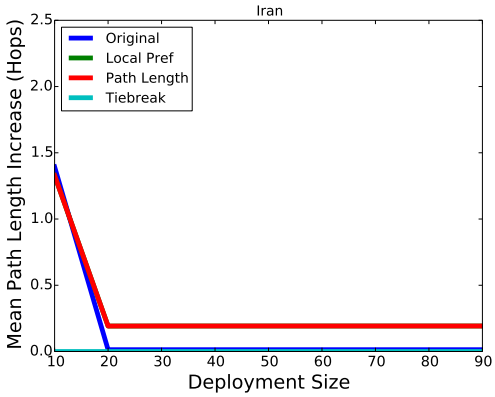


(a) Brazil

(b) China

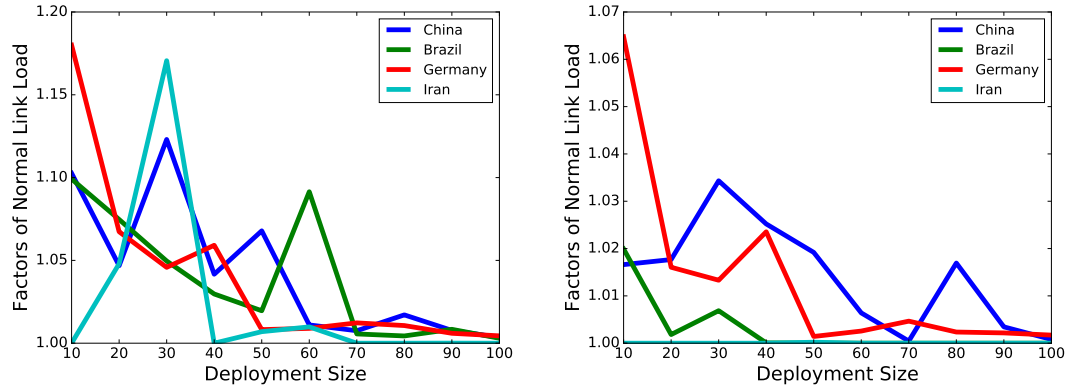


(c) Germany



(d) Iran

Figure 7.6: Mean AS level path length increase for our collection of RAD strategies operating against the targeted deployment. In all cases the Tiebreak strategy sees zero path length increase.



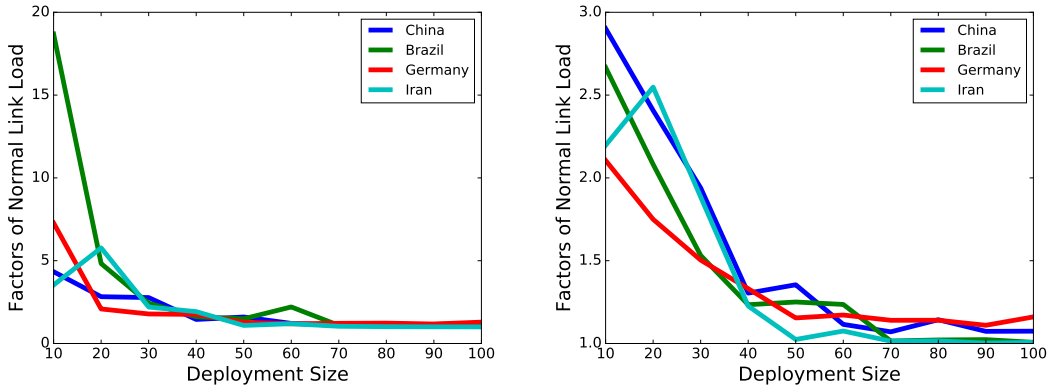
(a) Original

(b) Local Pref

(c) Path Length

(d) Tiebreak

Figure 7.7: Median increase in link load when considering links which must carry additional traffic as a result of resistor reaction to a global deployment. With the exception of the original RAD attack, no strategy results in more than a single digit percentage increase.



(a) Original

(b) Local Pref

(c) Path Length

(d) Tiebreak

Figure 7.8: The 90th percentile of link increases for our resistors employing various strategies. These increases result from reaction to a global deployment.

Increased traffic load across a subset of links in the topology is an unavoidable consequence of altering the route decision making process. However, the relative magnitude of those increases can be mitigated by using lower cost strategies. Figure 7.7 shows the median relative link load increase, and Figure 7.8 shows the 90th percentile of relative link load increase as a fraction of normal load, only links experiencing a load increase were considered. Note that, unlike other plots in this section, these graphs are clustered via resistor strategy rather than resistor. The original RAD attack in all cases results in the largest link load increases of any strategy, a natural result of utilizing customer links for transit purposes. The other three strategies result in, with one exception, a median relative link increase that never exceeds a 5% increase in link load, meaning that the majority of links are likely not impacted from a performance perspective. The 90th percentile shows that, for smaller deployments, some links experience a doubling or tripling of traffic load. Recent measurements [82] have shown that connection points between ASes are generally less than 50% utilized. It is likely in some extreme scenarios exist where additional infrastructure would need to be built, but those are the overwhelming minority of cases. The actual monetary cost to the resistor to build this infrastructure is dependent on a large number of factors and outside the scope of this paper. One phenomena that might mitigate these financial costs is the existence of large amounts of fiber connections which are either under utilized or not currently utilized at all, so called Dark Fiber [83], which could be tapped to provide additional capacity. Additionally, many of the links which see large increases exist inside of Internet Exchange Points, or IXPs, where adding capacity, from a technical perspective, is very straightforward.

7.4 Deployer Costs

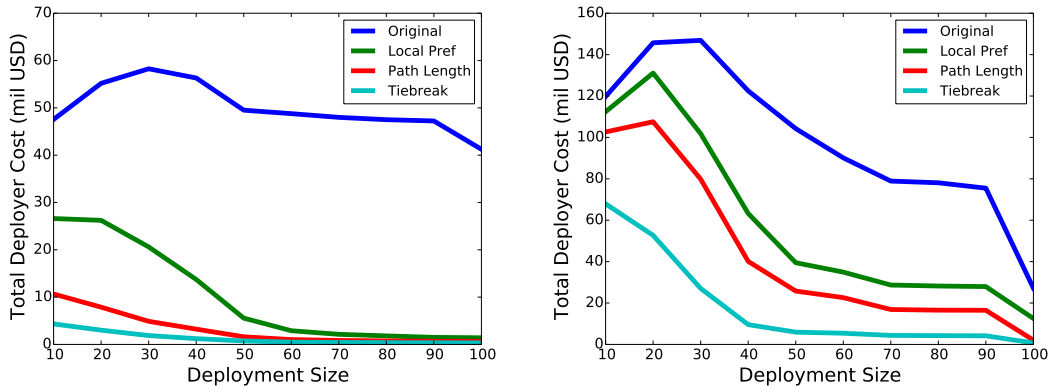
In order for TMBs to be useful, deployers must be transit providers, an AS that generates revenue by carrying traffic to and from their customer ASes. While transit revenue is obviously not the only way ASes generate revenue in this work we focus exclusively on transit revenue. As covered in Section 7.2, the amount of revenue a transit AS generates from a customer is proportional to the volume of traffic handled for that customer. Since the RAD attack directs the resistor's data away from paths that cross deployers, by extension the attack also directs the revenue away from the deployers.

This observation opens up a new strategy for the resistor. While the original RAD attack model focused on directly undermining the availability of TMBs, our new attack instead focuses on inflicting economic damage on deployers with the goal of incentivizing them to remove the TMBs. In this attack model our resistor is, in the short term, willing to allow traffic to cross TMBs. The resistor instead focuses on the long term goal of inflicting economic losses so severe on deployers that the originator lacks the resources to defray these losses, and the only rational decision deployers can make is to remove the TMBs. The rest of this section will measure how large the economic harm is from the traffic deflection results presented in Section 7.3, expand upon those attacks by taking steps to influence traffic going *to* the resistor, and examine the added incentives to defect from the ranks of deployers.

7.4.1 Direct Costs of Deployment

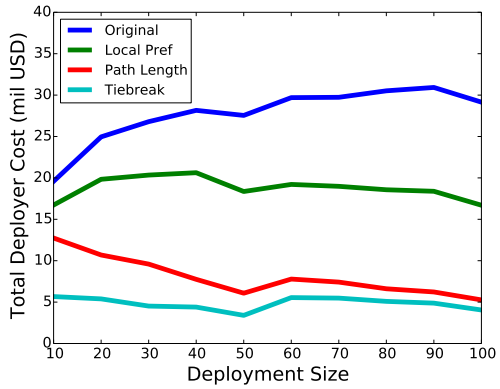
Using our simulator, we examine the RAD attacks of Section 7.3 in terms of deployer revenue losses. To quantify these losses, we compute the amount of revenue made by deployers before and after resistors attempt to route around them. In keeping with the concept that the resistor will not disconnect itself from any destination, during all of our simulations if a resistor can not find a clean path utilizing its chosen RAD strategy, a tainted path is utilized instead. The originator will need to reimburse the deployers for revenue lost as a result of resistor actions, and the sum of these reimbursements represents the originator's annual *cost of deployment*.

Figure 7.9 shows the originator's cost of deployment for targeted deployments against our resistors, and Figure 7.10 shows the cost of global deployments. We see that the *annual* economic damage routing capable adversaries can inflict, even using low cost RAD strategies, ranges from hundreds of thousands of dollars to tens of millions of dollars. To put those numbers in perspective, consider the United States Broadcasting Board of Governor's Internet Anti-Censorship division, the branch of the US government with the expressed task of acting as the originator for systems like decoy routing. The *total* budget in 2015 for this organization was 15 million dollars [84]. Thus, depending on resistor strategy, the annual cost of decoy routing deployment would equal or exceed its total operating budget.

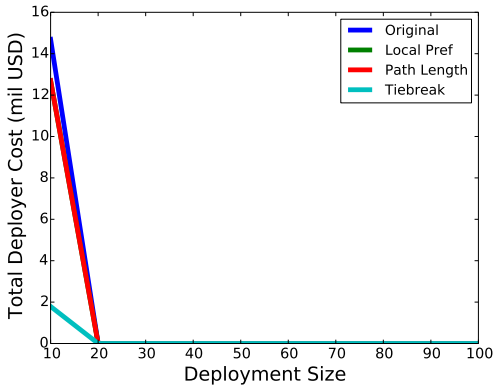


(a) Brazil

(b) China

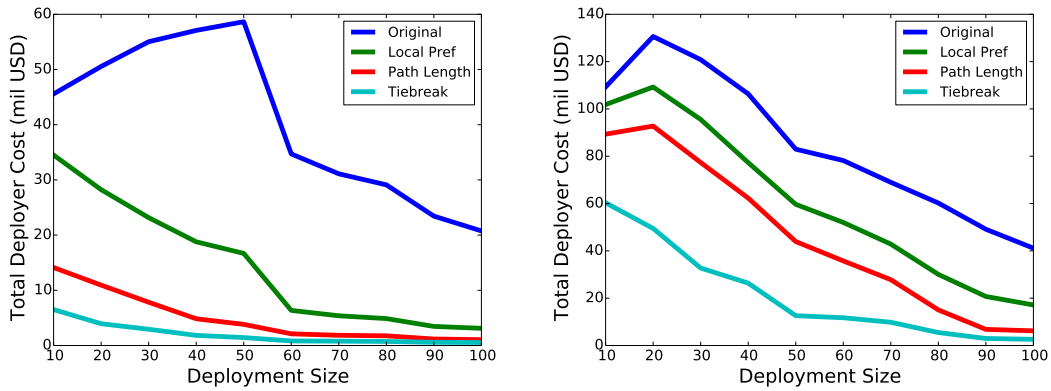


(c) Germany



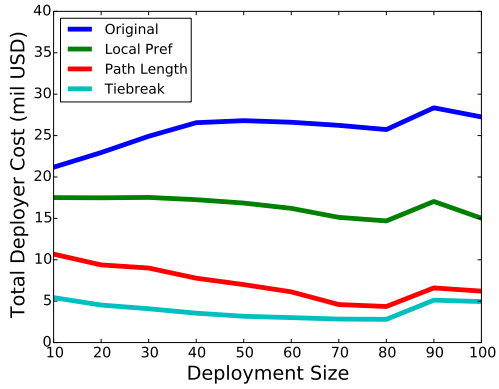
(d) Iran

Figure 7.9: The cost of deployments targeted against specific resistors as a function of deployment size. This is an annual cost that would have to be paid by the originator to defray lost revenue experienced by deployers.

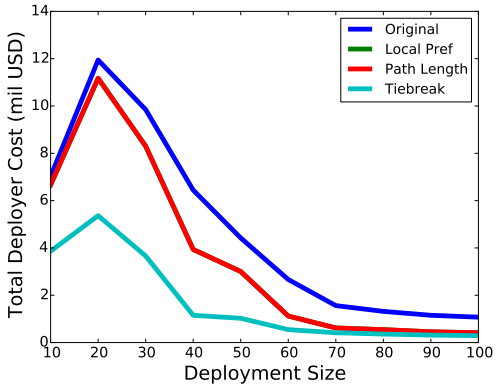


(a) Brazil

(b) China

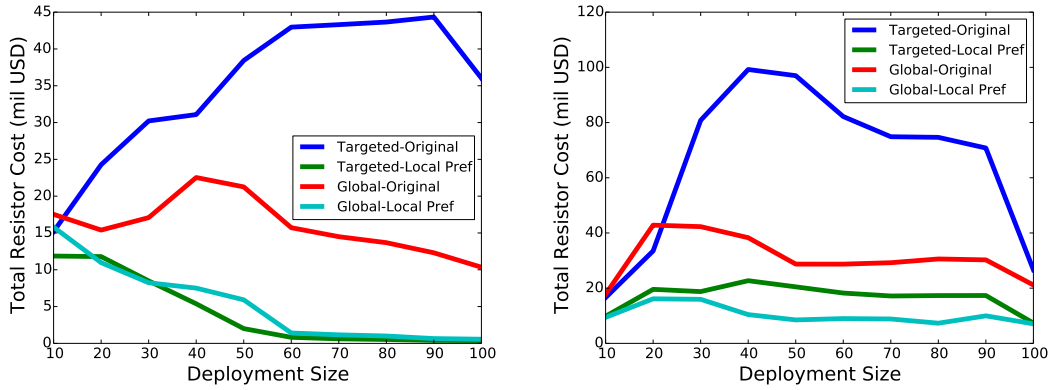


(c) Germany



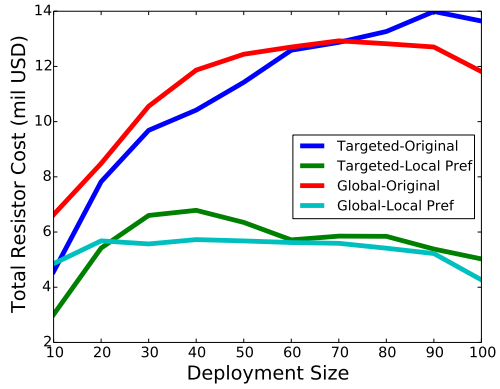
(d) Iran

Figure 7.10: The annual cost of a deployment maximizing global coverage of traffic when reacted to by each of the resistors.

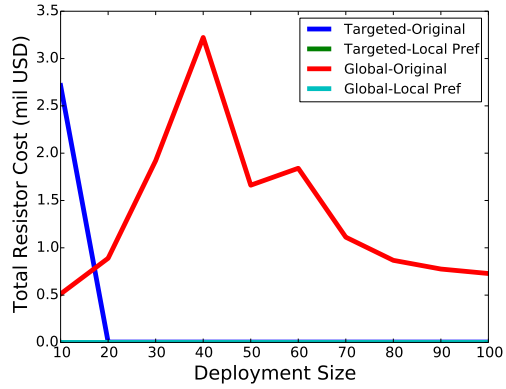


(a) Brazil

(b) China



(c) Germany



(d) Iran

Figure 7.11: The annual costs experienced by resistors when using the original RAD attack and the Local Preference strategy. These costs arise from having to reimburse resistor ASes for transiting traffic for their providers and added transit costs that come from migrating from customer learned paths to provider learned paths.

As would be expected from the results of Section 7.3.2, more aggressive RAD strategies result in higher costs of deployment. For example, when looking at a targeted deployer being resisted by China (Figure 7.9b), switching from a Path Length strategy to a Local Preference strategy results in an average of 38% increase in the cost of deployment and shifting from Local Preference to the original RAD strategy gives an average improvement of 81%. Other trends from traffic deflection graphs presented in Section 7.3 apply to these graphs as well. For example, Brazil sees a massive increase in capabilities when switching to the original RAD attack.

The raw volume of traffic a resistor generates, in addition to what fraction of that can be deflected, now plays a role in how well our resistors can influence the cost of deployment. For example, looking back to Figure 7.3, Germany can deflect a higher percentage of its tainted traffic away from deployers. However, when we compare the resulting costs of deployment for Brazil, Figure 7.9a, to Germany, Figure 7.9c, we see that for many sizes of deployment, the cost of deploying against Brazil is actually larger than the cost of deploying against Germany. While Germany can find more clean paths, Brazil's directly controlled volume of traffic is larger than Germany's, meaning that the clean paths it finds can be more economically damaging to deployers. Additionally, Brazil's customers tend to have fewer choices compared to ASes that are customers of Germany, meaning that while Germany has a large customer cone, the smaller customer cone of Brazil is more likely to utilize Brazil's post-RAD attack routes.

An interesting phenomena to note is that after a certain size, larger deployments are actually *less* expensive than smaller deployments. This is a direct result of resistors being less able to find clean paths. The extreme case of this is the targeted deployment against Iran (Figure 7.9d) where the RAD related costs fall to zero. However, the willingness to preferentially route using clean paths results in a Prisoner's dilemma like game, creating strong incentives for deployers to remove TMBs. This game will be introduced in Section 7.4.3.

Two of the RAD attack strategies, the original RAD attack and the Local Preference, inflict economic damages on the resistor as well as the deployers. Both of the these strategies can force resistor ASes to utilize provider routes over customer routes. In addition, the original RAD attack can force customer ASes to provide transit for their providers. We can measure these costs as well utilizing our simulator. Figure 7.11 shows

these resistor costs for both strategies against targeted and global deployments. We can see from the Figures that the original RAD attack is strictly more costly than a Local Preference strategy.

7.4.2 Impacting Incoming Traffic

The costs of deployment presented in Figure 7.9 and Figure 7.10 were the result of resistors controlling traffic *leaving* their ASes, but what about the traffic addressed *to* their ASes? While BGP places the final decision about path selection into the hands of the routers forwarding traffic, there are techniques that give the destination AS a measure of control over what routes are used to reach their IP blocks. BGP hole punching is one such traffic engineering technique. BGP allows ASes to advertise sub-blocks of an already advertised block of IP addresses, treating these sub-blocks as a different destination. We call these routes to sub-blocks of existing destinations *hole punched routes*. When a router attempts forward a packet, it will use the path for the most specific (i.e. smallest) IP block that contains the destination IP address. In this way operators can specify different path properties for a sub-block of IP addresses through the use of hole punched routes.

Resistors can adapt this traffic engineering technique to cause other ASes to avoid sending traffic bound for resistors through tainted ASes when able. The basic idea is for resistors to generate and propagate a set of hole punched routes which are guaranteed to not cross any deployers. When an AS outside of a resistor forwards a packet to a resistor, if the AS has one of these hole punched routes, it will always use it, causing traffic that might have once traveled over a tainted path to always use a clean path.

To do this, resistor ASes partition each block of IP addresses into two sub-blocks. The resistor then advertises three paths: the original path and a path for each of the two sub blocks. The original path will spread through the Internet normally and will be used in all cases when ASes do not have a hole punched path. This ensures that the resistor faces *no connectivity loss*. The hole punched paths differ from the original path in one key way: when the resistor advertises the path after his ASN, he adds all known deployer ASes to the AS path, then adds his ASN again. The resistor can do this since no AS path integrity protection is provided in BGP. The hole punched paths will propagate through the network, but they will be rejected by deployer AS routers

as a result of BGP loop detection. This causes the hole punched paths to grow *around* deployer ASes. The fact that the deployer ASes appear in the path is irrelevant for packet forwarding, since they appear after the destination AS. This technique ensures that if an AS has at least one clean path to a resistor, it will always have a clean hole punched route. We call this technique *Fraudulent Route Reverse Poisoning*, or FRRP.

We re-ran our experiments with the addition of FRRP for three deployment strategies: the original RAD attack, Local Preference, and Path Length. The Tiebreak strategy was not used in combination with FRRP since any resistor using the Tiebreak strategy has a goal of ensuring no increase in AS level path lengths. As shown in Figure 7.12 and Figure 7.13 FRRP can result in a small increase in the AS level path length for routes traveling into the resistor.

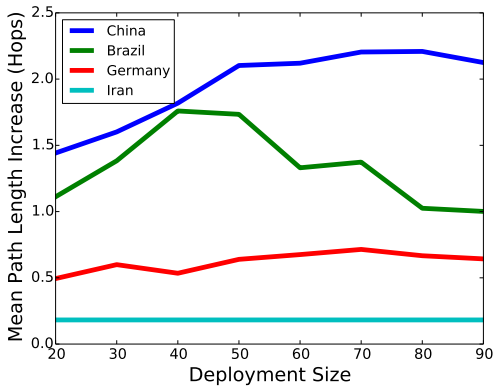


Figure 7.12: The mean increase in path length *into* the resistors as a result of reverse poisoning against a targeted deployment.

Figure 7.14 and Figure 7.15 show the costs of deployment for targeted and global deployments respectively where FRRP is employed in addition to a RAD strategy. FRRP increases the cost of deployment by more than 40 percent in all scenarios. In many cases the improvements are much larger, for example China can roughly double the economic losses inflicted on deployers by utilizing FRRP.

FRRP is made possible by the ability of the resistor to falsify the existence of deployers in the AS level path. This falsification can be done because of the lack of AS path integrity protection provided by BGP. Even the Resource Public Key Infrastructure (RPKI) [85], a partially deployed system attempting to provide integrity for route

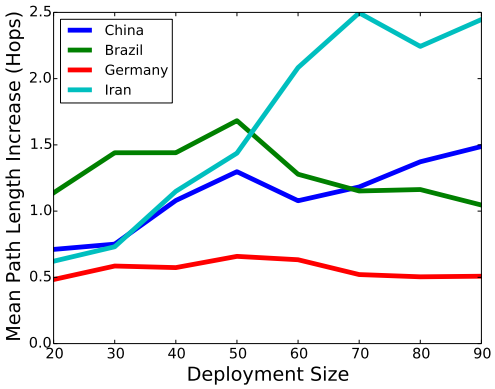
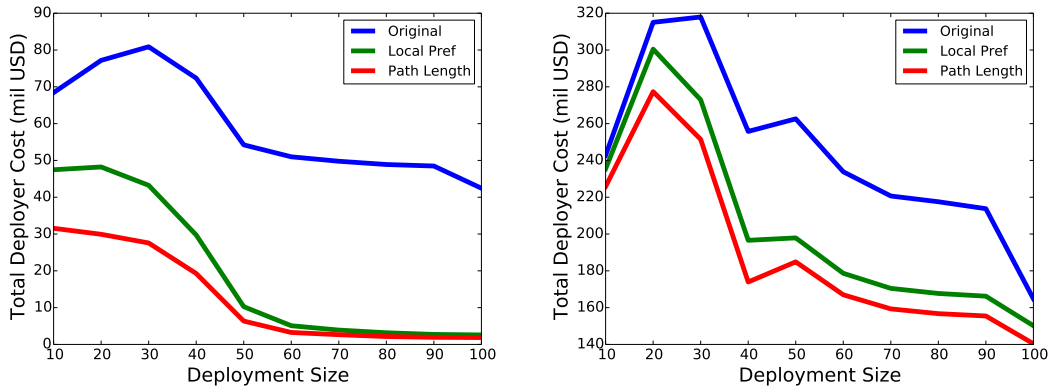


Figure 7.13: The mean increase in path length for in-bound paths to the resistor as a result of reverse poisoning against a global deployment.

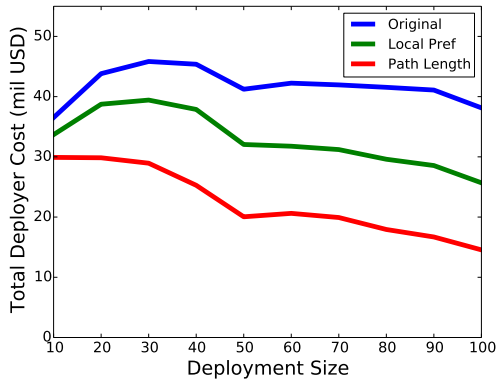
origination, provides no protection for the integrity of the path itself since the last ASN, the originator, is the correct ASN. BGPSec [86] would provide AS level path integrity, preventing the FRRP routes from being considered valid. In this case, resistors could not rely on loop detection at the deployers to guarantee that a hole punched path does not include any deployers, defeating the scheme. While BGPSec is not currently deployed, and thus not an immediate issue for a resistor, we describe a weaker version of reverse poisoning, *Selective Advertisement Reverse Poisoning*, or SelARP, which does not require falsifying path information, and would consequently still function if BGPSec were to be deployed at some future date.

SelARP takes advantage of how BGP paths are propagated through the topology. In BGP, advertising a path to a single neighbor does not guarantee that the route will be propagated to every AS in a topology. Deployers can therefore, by carefully selecting which neighbors they advertise the hole punched routes to, avoid allowing hole punched routes to propagate to deployers. Paradoxically, in some instances advertising a hole punched path to a neighbor which would result in a deployer later in the topology seeing the route is the optimal decision for the resistor, if prior to encountering the deployer the hole punched path manages to steal away traffic from a *different* deployer. Because the hole punched paths are not advertised to all neighbors, the guarantee that we had previously of an AS which has at least one clean path to a resistor always having at least one clean hole punched route no longer applies.

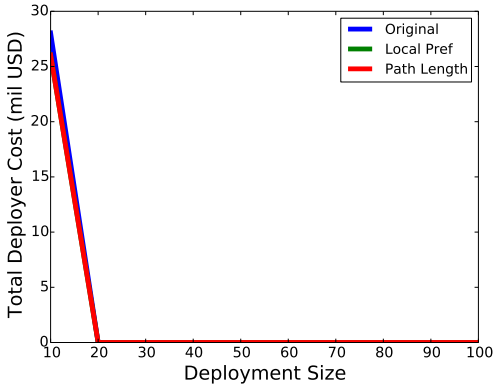


(a) Brazil

(b) China

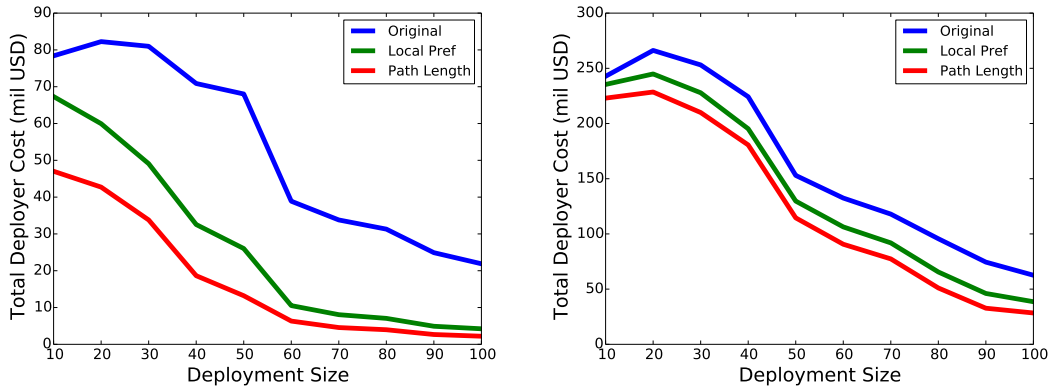


(c) Germany



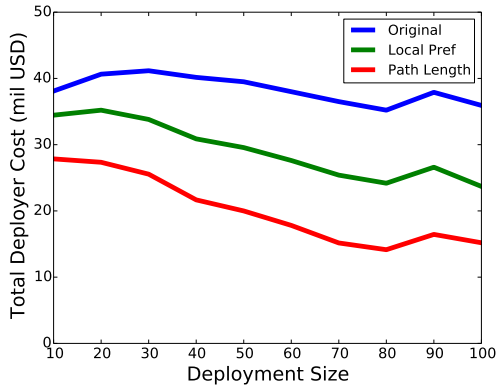
(d) Iran

Figure 7.14: The annual cost of targeted deployments against our resistors when both a RAD strategy and FRRP is used. Tiebreak strategy is not shown since FRRP poisoning slightly increases path length, and so is incompatible with a Tiebreak strategy.

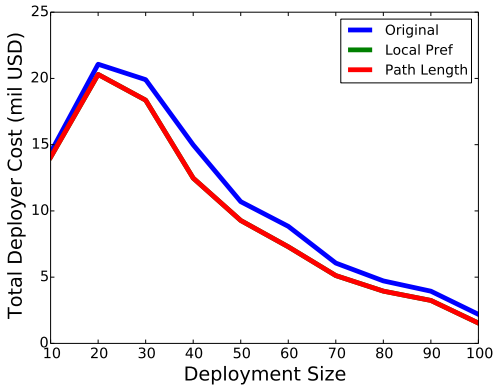


(a) Brazil

(b) China



(c) Germany



(d) Iran

Figure 7.15: The annual cost of a global deployment when resisted by both a RAD attack and FRRP.

Data: Set of neighbors
Result: Set of neighbors to advertise hole punch
$score \leftarrow 0$;
$AdvertiseSet \leftarrow \{\}$;
$Unadvertised \leftarrow Neighbors$;
repeat
  $roundGain \leftarrow 0$;
  $roundChoice \leftarrow$;
  forall $candidates \in Unadvertised$ **do**
    advertise hole punched route to *candidate*;
    $tempScore \leftarrow$ current deployer losses;
    if *tempScore* - *score* > *roundGain* **then**
      $roundGain \leftarrow tempScore - score$;
      $roundChoice \leftarrow candidate$;
    end
    withdraw hole punched route to *candidate*;
  end
  advertise hole punched route to *roundChoice*;
  $score \leftarrow score + roundGain$;
  $AdvertiseSet \leftarrow AdvertiseSet \cup roundChoice$;
  $Unadvertised \leftarrow Unadvertised \setminus roundChoice$;
until $roundGain = 0$;

Algorithm 1: Greedy algorithm for selecting which neighbors to advertise the hole punched route to.

The problem of selecting which subset of neighbors to advertise the hole punched path to such that it maximizes the amount of return traffic avoiding deployers appears to reduce to the set covering problem, which is NP-hard. A greedy estimator can be built to provide a lower performance bound. This greedy estimator considers every neighbor that does not have a hole punched route advertised to it, tentatively advertises the hole punched path to it, and measures the number of ASes (weighted by IP addresses) that have a clean hole punched route after the path is propagated by BGP. If at least one neighbor increases the number of ASes with a clean hole-punched path, the neighbor

which results in the most such ASes is then added to the set of neighbors who receive the hole punched path. The topology is then permanently updated with the hole punched route advertised to this neighbor. The process is repeated until no more such neighbors can be found. This algorithm is given in Algorithm 7.4.2.

We evaluated the ability of SelARP to influence traffic in the face of a targeted deployment. While SelARP is weaker than FRRP, it still has the ability to cause traffic which would previously traverse a deployer to instead travel a clean path. The CDF of the fraction of incoming traffic, previously deflected via FRRP, which each resistor AS manages to still deflect away from deployers by SelARP is shown fin Figure 7.16 for our sample resistors. As can be seen, for the three tested resistors, the majority of member ASes are still able to deflect some amount of incoming traffic using SelARP. For Brazil and Germany nearly 20% of ASes have identical reverse poisoning capabilities when using SelARP as they do while utilizing FRRP. More refined methods for selecting which neighbors to advertise hole punched routes to, along with additional techniques such as making some hole punched routes appear less attractive by artificially inflating path length might improve these results and is left for future work.

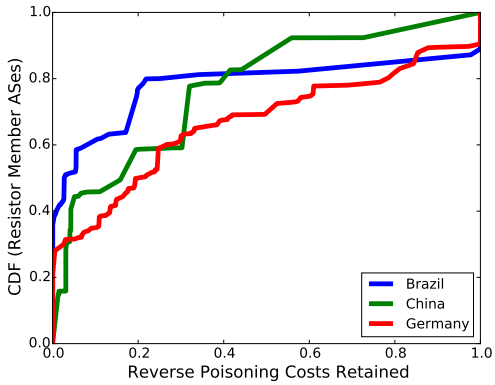


Figure 7.16: CDF of the fraction of reverse poisoning costs that are retained when SelARP is used. This test is done against targeted deployments of 30 ASes.

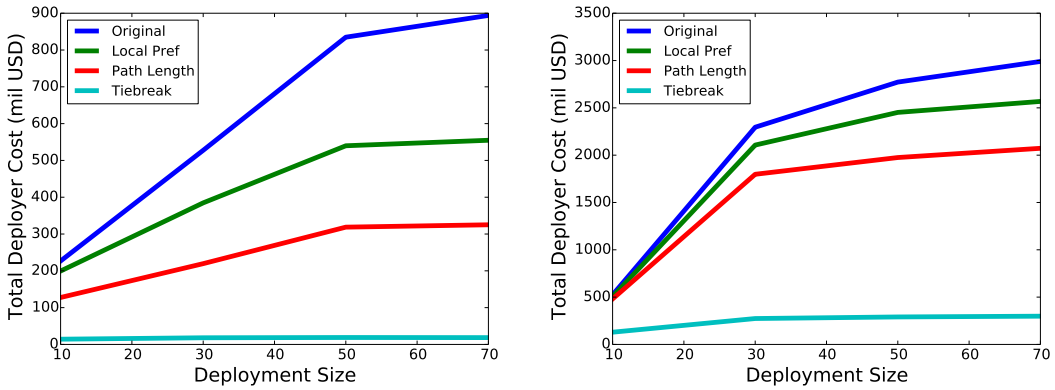
7.4.3 Deployer Defection

If we revisit our traffic deflection graphs, Figure 7.3 and Figure 7.4, we see that a large fraction of traffic has no clean path after resistor reaction. An interesting scenario occurs

when there are multiple deployers who present the resistor with a path to one of these destinations. One of those ASes will have the best (i.e. utilized) path, however resistors prefer clean paths over tainted ones. If one of the deployers who is providing a sub-optimal path to the destination removed its TMBs, an action referred to as *defection*, its path would now be the preferred path by virtue of being the only clean one. By doing this, a defecting deployer steals billable traffic away from other deployers. This means there are actually not one, but two costs experienced by deployers when they agree to host TMBs. First, there is the previously discussed loss in billable traffic that results from paths they once provided migrating to clean ASes. Second, there is an opportunity cost associated with the billable traffic they could gain by defecting and replacing other deployers as a best path provider, what we term *defection traffic*.

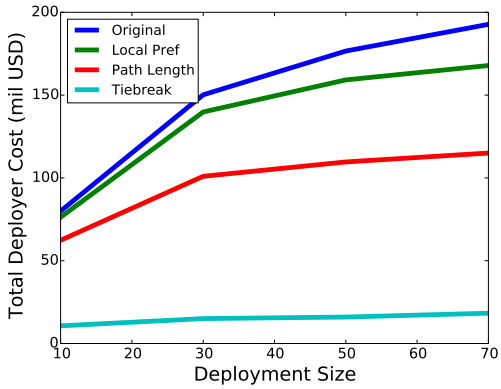
When deployers force resistors to defray these opportunity costs, the cost of deployment increases by more than an order of magnitude. Figure 7.17 shows the measured cost of targeted deployment when both the direct cost and the defection opportunity costs are considered. The original RAD attack, Local Preference, and Path Length strategies, all augmented with FRRP, are considered, along with the Tiebreak strategy using no FRRP. These four combinations represent a range of resistor costs ranging from the willingness to invest several million dollars to compensate member ASes down to no added cost.

The original RAD attack and Local Preference strategy (both augmented with FRRP) represent the strongest resistors. These resistors are exceptionally powerful, inflicting losses in the hundreds of millions of dollars, in the case of Brazil approaching 1 billion dollars, and in the case of China exceeding 2 billion dollars of annual cost to the originator. This amount of economic harm is large, even when considering powerful originators such as spy agencies. For example, China would be able to require the NSA to pay out approximately 20% - 30% of its total budget [87] to cover the cost of a large deployment. The financially free strategies can also inflict substantial losses. For all resistors, the Path Length with FRRP strategy yields costs of deployment at or exceeding roughly 100 million dollars annually. The free attack of simply Tiebreaking based on path cleanness imposes annual costs of deployment ranging from 15 million dollars for Germany to 290 million dollars for China, in all cases equaling or exceeding the total budget of existing public and private free speech advocacy organizations.

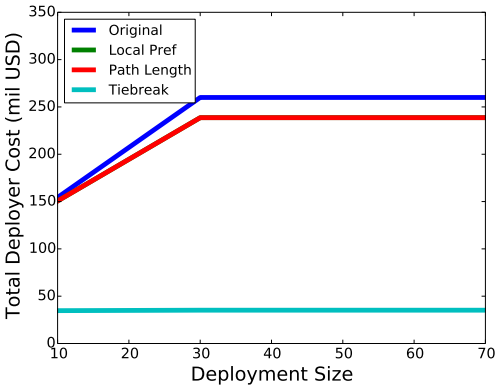


(a) Brazil

(b) China



(c) Germany



(d) Iran

Figure 7.17: The cost of targeted deployments, including defection costs, when a RAD attack and FRRP is utilized. Note that unlike the prior costs, when defection is considered, Iran is a viable resistor.

Two key differences should be noted when comparing the costs of deployment including defection with those ignoring defection. First, larger deployments are no longer less costly compared to smaller deployments. This is a result of the incentive for deployers to defect *increasing* as more and more resistor traffic needs a clean path. The second phenomenon, which is closely linked to the first, is the emergence of Iran, shown in Figure 7.17d, as a strong resistor. As covered previously, the targeted deployment rapidly surrounds Iran with deployers, which means that all RAD attacks fail. However it also means that nearly *all* of Iran's traffic is up for grabs for any adjacent AS that elects to defect. This demonstrates the difficulty faced by the originator strategy of "blockading" a target. While establishing the blockade may be simple, maintaining it is costly.

7.5 Discussion

7.5.1 Alternative Deployment Strategies

Instead of a nation state acting as a resistor, it could alternatively act as a deployer. For example a government could have all ASes inside of a country deploy TMBs. This would of course trivially prevent clean paths from being found to any destination inside the country. On the surface this appears to be an effective deployment strategy, as resistors can not find alternative paths to any destinations inside the nation. However, the interconnected nature of the Internet means that quite often a large fraction of traffic flowing through a transit AS is bound for a nation *other* than the one the transit AS resides in. While it is true that all of the traffic destined for delivery inside the deployer nation state can not be diverted away from deployers without connectivity loss, traffic passing through the deployer state can. Therefore deployer nations risk the loss of this traffic due to resistor action. Figure 7.18 shows a CDF of the fraction of transit traffic flowing through transit ASes in our simulator that is bound for a destination in a different nation than the one the transit AS resides in. As can been seen, for approximately 80% of transit ASes more than a third of all transit traffic is bound for a destination abroad. Fully two fifths of transit ASes see half or more of their traffic heading to foreign destinations.

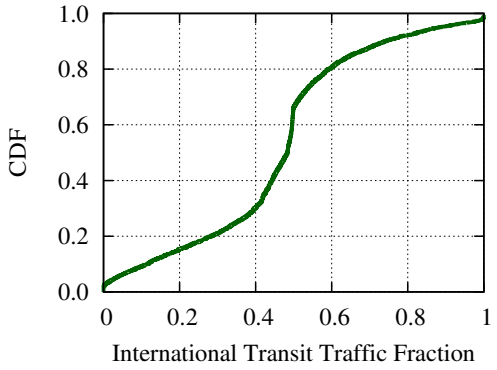


Figure 7.18: A CDF of the fraction of transit traffic bound for a destination that is international relative to the transit AS.

7.5.2 When Originators are nation states.

In some of the motivating examples we highlighted TMBs that conduct surveillance for a nation state. This scenario presents multiple additional challenges to our resistors. To begin with, the budgets of spy agencies, as covered in Section 7.4, are far larger than that of NGOs, on the order of billions of dollars. However, some resistors, for example China, can generate deployment costs on the order of several billion dollars annually against global deployments. It should also be noted that if multiple resistors launch economically focused RAD attacks the losses inflicted on deployers only increase. Given that one of our motivating examples is a set of alleged TMBs deployed by the NSA, it is interesting to consider a deployment scenario similar to the one just discussed, where the United States is ringed with TMBs. In this scenario, rather than focus on a single nation state resisting we consider a coalition of the top 5% of ASes, ranked by degree of connectivity, acting as a resistor. TMBs are deployed to all ASes which primarily operate in the United States that have a link to an AS which resides in another nation. The deployment costs, both direct and those including defection costs are shown in Table 7.3. The first thing to note is that even though no clean paths exist into the United States, there are still direct losses on the order of half a billion dollars annually as a result of some international transit traffic no longer traversing through the United States. In addition, we see that defection pressure increases deployment costs by nearly an order of magnitude for all strategies except Tiebreak.

	Direct Costs	Defection Costs
Legacy	529.7 mil USD	5017 mil USD
Local Pref	496.5 mil USD	4150 mil USD
Path Len	422.7 mil USD	3036 mil USD
Tiebreak	174.2 mil USD	871 mil USD

Table 7.3: Direct and opportunity costs for ASes belonging to the United States when a "ring" deployment around the US is resisted by a coalition of 5% of all ASes.

When the originator is a nation state other complications for our resistor's attack model exist. For example there is the question of whether or not deployers are allowed to act as rational economic entities. If the deployers are legally compelled to host TMBs without compensation then the originator would not bear any of the cost of our resistor's actions. However, it is important to note that our resistor's actions are still functional, as they provide a strong economic incentive, which did not exist prior to the resistor actions, for the deployers to push back against the forced deployment of TMBs to their network, which is the resistor's goal in the first place. An additional complication might be the surreptitious deployment of TMBs, in such a case a "deployer" may be unaware of the location or even presence of TMBs on its network, making defection improbable – although with resistor detection and action, high enough opportunity cost may motivate the deployer AS to discover these facts.

7.5.3 Finding deployers of TMBs.

We note that for several examples of TMBs, it is essentially trivial to identify deploying ASes. Our prior work in Chapter 6, showed how, in the original RAD attack, deployers can be identified by using the decoy routing protocol itself: the efficacy of the protocol depends on users being able to signal a covert destination to the TMB. Thus, a given AS path can be tested using this signal; if the covert destination is retrieved, then the path includes a deployer. Repeating this process with each AS path available to the resistor allows the resistor to identify the deployers using an intersection attack. Given a known keyword, similar processes have been used to identify ASes that manipulate traffic by keyword filtering [88], ad injection [89], and DNS injection [90]. Work has even recently been presented which has had some amount of success detecting the QUANTUM INSERT infrastructure [91].

More subtle traffic manipulation such as traffic shaping and protocol differentiation might require probabilistic inferences by the resistor [92, 93], leading to the possibility of classification errors. In this case, we note that since the new attacks we discuss have little cost to the resistor, *false positives have limited consequence.* Let C be the cost our attack incurs on a deployer, α be the true positive rate, and ψ be the false positive rate. Then a TMB deployer will have expected cost αC under our attack, and a non-deployer will have expected cost ψC. Thus as long as $\alpha > \psi$ the attacks will disincentivize deployment; however, compared to manipulations that can be detected perfectly, the damages will be reduced by a factor of $(\alpha - \psi)$. Fully passive surveillance systems present the most challenging scenario for our resistor, however non-technological detection, for example through disclosures similar to those of Edward Snowden could be sufficient for resistors to launch a RAD attack against all member ASes of a given nation state to place pressure on the host government.

7.5.4 TMBs Which Require Total Disconnection

Readers familiar with Decoy Routers will have observed that they function so long as *any* tainted path exists from a given resistor. In this case our economic resistor will return to the policy of refusing to send any traffic to destinations which lack a clean path, however the economic resistor's position is strengthened via economic pressure on deployers. By increasing the cost on the originator, whole more than likely has a fixed budget to spend on TMB deployment, the resistor can reduce the size of deployment the originator can field. This in turn means that there will be fewer end destinations the resistor will need to cut itself off from when preventing any traffic from utilizing tainted paths.

7.5.5 Modeling limitations.

We note that there are some limitations to our cost model, which may tend to overstate the opportunity costs imposed by defection. Additionally, variations in pricing and contractual obligations to other organizations could also complicate the decision to defect. Without more detailed information about the pricing policies and exact traffic volumes

of individual ISPs, it is difficult to measure the impact of a resistor's actions with perfect accuracy. The models put forward in this paper exist to illustrate the concept of an economically based attack against availability rather predict exact losses to the last dollar.

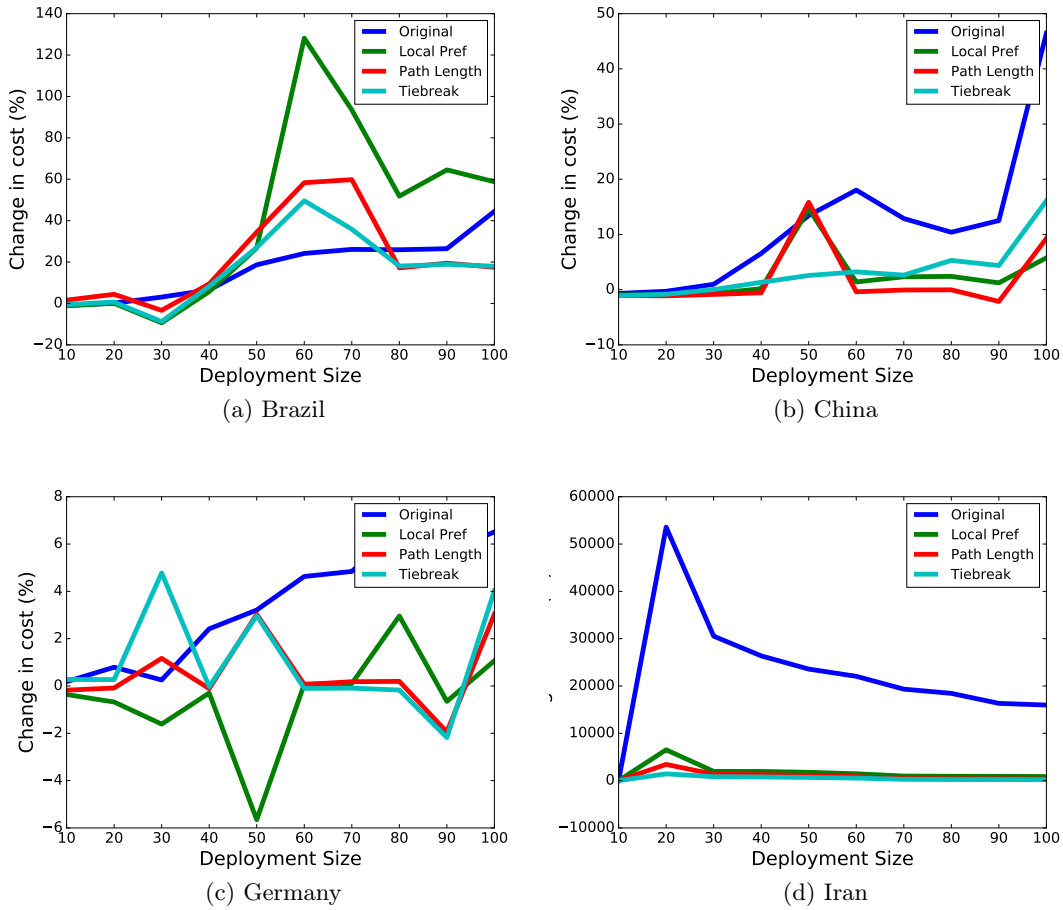
The other limitation of our cost model is the fact that only a single round of deployer incentives and defection is examined. Our defection costs present a maximum opportunity cost, as it assumes in each case that only one AS elects to defect. If two ASes simultaneously elected to defect it is possible that the amount of additional transit traffic they attract is lower than either expects; a result of the two ASes splitting the gains. However the opposite direction is also true, two or more deployers might form a *defection coalition*, creating clean paths which previously had multiple deployers. In this way the single defector scenario is not necessarily a worst-case scenario. An interesting question is what are originator costs when multiple rounds of deployment and defection are conducted? Additionally, preferential routing by our resistor could incentivize other ASes to build links to resistor ASes in an effort to capture traffic from the resistor which currently has no clean path. We feel that examining these scenarios is an exciting direction for future work.

Another complication that can result in errors in simulated paths is the existence of peering links between ASes which are difficult to detect from the outside. In order to test that our model is insensitive to the existence of such links we tested our attacks on topologies with additional peering links added to the topology. We found that in the majority of cases the addition of more peering links made our routing capable adversary stronger. This is shown in Figure 7.19 and Figure 7.20. We elected to use only the observable AS topology since it resulted in the weakest adversarial model.

7.5.6 Detection of reverse poisoning.

We note that as described, the reverse poisoning attack can be detected by an AS that monitors all incoming route advertisements. In principle, it might be possible to develop a routing policy that detects this and counters the poisoning by re-advertising the aggregate route for each sub-block. This policy would carry substantial risks of introducing routing cycles and other instabilities in the BGP infrastructure, and thus does not seem to be carried out in practice. Moreover, SelARP would be far harder

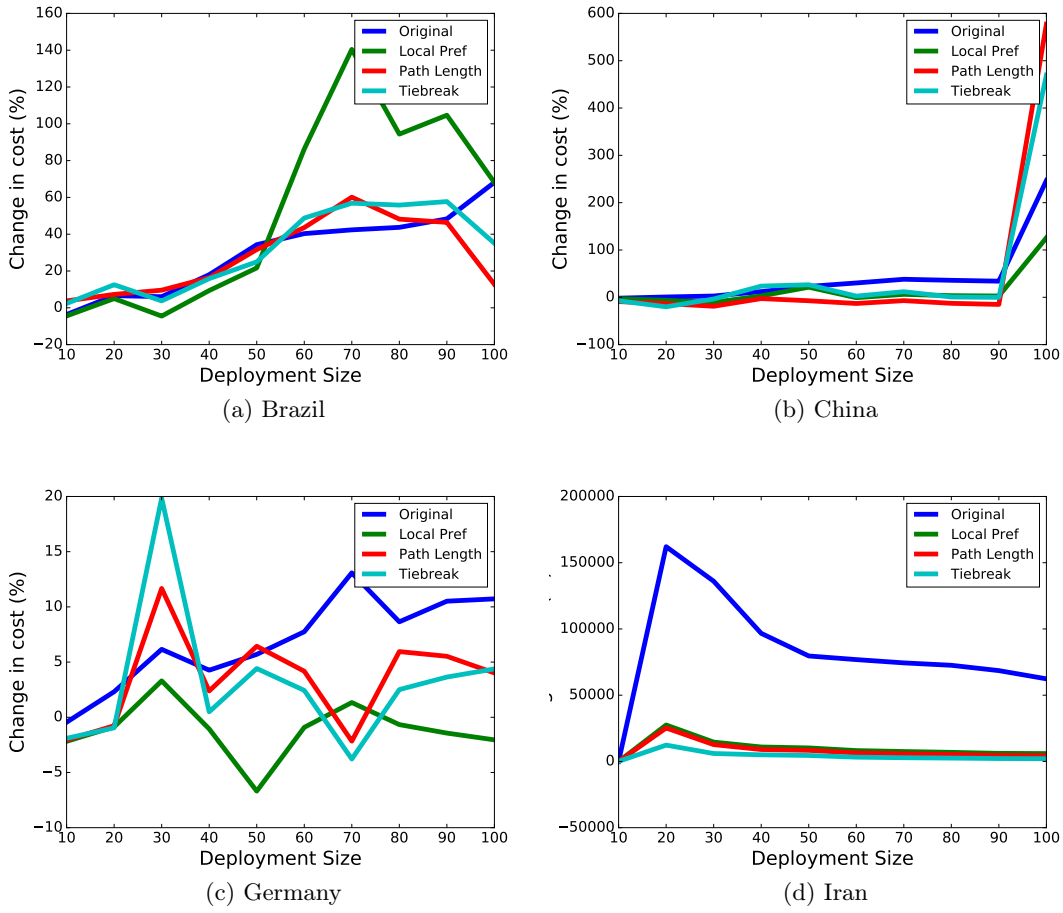
to detect. Additionally, ASes detecting non-lying reverse poisoning could only react to it by falsely claiming they too have the hole punched routes, something they would be unable to accomplish in the presence of BGPSec.



(a) Brazil (b) China

(c) Germany (d) Iran

Figure 7.19: Change in cost of deployment when our topology has an additional 10% of peering links added. Peering links are difficult to detect and may be missing from our AS topology. By adding additional peering links in nearly all cases our resistor is stronger.



(a) Brazil

(b) China

(c) Germany

(d) Iran

Figure 7.20: Chane in the cost of deployment when our topology has an added 30% of peering links. Again, our resistors are stronger as a result of the added links.

Chapter 8

Future Work and Final Remarks

8.1 Future Work

In this section we will discuss future directions for this research.

8.1.1 Cascading BGP Failures

There is an interesting connection between the attacks on router stability in Chapter 4 and the reaction of BGP to such instability in Chapter 5. The back pressure attack developed in Section 5.1.2 causes victim routers to crash by presenting the router with large numbers of BGP withdrawals and advertisements. When routers fail as a result of this attack, a large number of BGP advertisements and withdrawals are generated by nearby routers, the very thing that caused the first router to crash. One can envision a scenario where the BGP messages that result from a successful attack stress a second set of routers enough to cause them to crash in turn. These routers failing would in turn result in yet another wave of BGP messages. This scenario is referred to as a *cascading failure*. Would it be possible for an adversary to cause a cascading failure? Conversely, could critical section of the BGP topology be located and hardened to prevent such attacks?

Interestingly, this attack actually works against itself. Each router failure both results in more BGP messages being generated while at the same time *reducing* the capability of the routing infrastructure to propagate those messages. This forces an adversary into a balancing act. Destabilizing routers too slowly will result in too few BGP messages to crash additional routers. Crashing routers too quickly will prevent new BGP messages from propagating, burning the attack out. Can an optimal rate of failure be found such that the chain reaction of the attack continues?

It is a mistake to assume that the routers will only fail and never recover. Routers will attempt to automatically restore BGP connections after a certain amount of time. This is again a mixed blessing for an adversary. On the one hand this means that the attack must be able to constantly crash victim routers after they recover. On the other, router recovery could result in another wave of BGP advertisements as routes that were once withdrawn are propagated through the network again.

An interesting scenario can occur when one takes into account the fact that there is a sizable (on the order of 3 minutes) delay in automatically restarting BGP sessions.

An adversary launches one of the previously developed attacks against a set of routers, call them A. Some routers from set A are connected to a second set of routers, call them B. As A fails, a surge of updates is generated, much like in the CXPST attacks, these updates overwhelm the computation capacity of B. This results in back pressure inside of B, which over time will result in router failure. Routers in B begin to fail as routers in A attempt to automatically restart. This means routers in A will be forced to handle withdrawal messages from set B along with advertisement messages from set A attempting to restart. This causes back pressure in set A, causing router failures to occur at the same time set B begins to restart, and the process repeats itself. This scenario is termed an *oscillating failure*, and would be an attacker's ideal goal, as the attack, once started, would be self-sustaining until operator intervention occurs. There are obvious technical hurdles for an adversary to overcome in order to launch this attack. The most prominent obstacle would be ensuring that the timing between router recovery and router failure is aligned. Is it possible for an adversary to engineer an oscillating failure scenario, and would such a failure mode be "stable"?

8.1.2 Defection Pressure Revisited

The analysis of defection pressure on deployers of TMBs in Chapter 7 focused only on a single round of interactions between the originator, deployers, and resistor. What happens if that time line is extended and multiple rounds of interaction are played? How would an iterative game impact defection pressure? Are there deployment strategies that are more optimal for the originator in an iterative deployment scenario? In addition, the game could be expanded to include one more player, the non-deployer ASes. The non-deployer ASes have economic motivations to take a variety of actions, including reacting to lost revenue as a result of propagating tainted paths and building new links to resistors in order to steal traffic away from defectors. How will the addition of this actor impact deployment costs that the originator will have to reimburse?

8.2 Conclusions

In this work we have explored the thesis that the current state of BGP, coupled with the Internet's extreme level of topological complexity, allows adversaries who can interact with BGP routers to degrade the availability properties of both the entire Internet routing infrastructure and other Internet scale distributed systems. We examined evidence of multiple ways adversaries could disrupt the functionality of large systems of BGP speakers, include the Internet as a whole. We also demonstrated how an AS can violate security properties of certain categories of distributed systems while still managing to accomplish their primary job of providing data delivery. There are two primary conclusions we can draw from this work.

First, the availability properties of the routing infrastructure of the Internet, properties many critical systems assume as a precondition, are attackable by dedicated adversaries. While BGP was designed with the goal of being resilient to failure, the actual result was a system that is resilient to *random and localized* failure, but exceptionally brittle to both *widespread* failures and *targeted* failures. This state of affairs results from a confluence of several factors including: the size of today's Internet, how operators elect to advertise the networks they own, the level of trust ASes place in information coming from other ASes, the low computational capacity and headroom of modern routing hardware, assumptions made by software engineers who implement BGP, and a routing protocol that has in its specification more than 100 "bolt on" fixes and extensions.

Both the security and networking community have largely put on blinders when it comes to these issues. The community notices accidental failures in the topology have wide ranging consequences beyond the localized area of the failure, but hand wave it away as a one time event. We see flaws in how BGP handles various scenarios which can lead to total failures of security models and ignore it by saying "Oh, it's BGP, everyone knows it is broken". We look at adversarial models which include adversaries willing to disrupt large portions of the Internet and write those threat models off as unrealistic. The truth of the mater is our inter-AS routing infrastructure was not designed with security in mind, and the result is a system that is ripe for disruption by willing and dedicated adversaries. Given the number of systems which we deem as vital to our day to day functioning as a nation that depend on the Internet, the laissez-faire approach

to the security of our routing infrastructure is reckless at best and courting disaster at worst.

Second, designers of distributed systems, when building their security model, need to take into account the fact that routing capable adversaries exist. To put it another way, system designers need to not assume that the routing choices made by ASes are agnostic to their presence or begin. This is especially true for both TMBs, since they interact closely with the routing infrastructure, and systems which would attract nation state level adversaries, for example censorship circumvention systems. Decoy routing systems are an excellent example of what happens when a system that meets *both* of these criteria ignores routing capable adversaries. The result is a large amount of research energy expended investigating deployment models which are fundamentally flawed. Given the limited resources that the censorship circumvention community has to drive deployment of functional real world solutions, moving forward we should ensure that the systems we propose and build take into account the routing infrastructure they operate over.

References

[1] BGP Reports. http://bgp.potaroo.net/.

[2] Earl Zmijewski. Reckless driving on the internet. Renesys Corp., http://www.renesys.com/blog/2009/02/the-flap-heard-around-the-worl.shtml, 2009.

[3] Andree Toonk. What caused today's internet hiccup. http://www.bgpmon.net/what-caused-todays-internet-hiccup/, 2014.

[4] Max Schuchard. Stormcaller Simulator. https://github.com/pendgaft/stormcaller.

[5] Max Schuchard. Nightwing Simulator. https://github.com/pendgaft/nightwing.

[6] CISCO Systems. CISCO Systems - Routers. http://www.cisco.com/en/US/products/hw/routers/index.html.

[7] Juniper Networks. Juniper Network Routing Solutions. http://www.juniper.net/us/en/products-services/routing/.

[8] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006. Updated by RFCs 6286, 6608, 6793.

[9] Network Working Group. RFC1997 - BGP Communities Attribute. http://tools.ietf.org/rfc/rfc1997.txt, August 1996.

[10] L. Gao and J. Rexford. Stable internet routing without global coordination. *IEEE/ACM Transactions on Networking (TON)*, 9(6):681–692, 2001.

[11] Alin C. Popescu, Brian J. Premore, and Todd Underwood. The anatomy of a leak: AS9121. Technical report, Renesys Corp., May 2005.

[12] James Cowie. China's 18-minute mystery. Renesys Corp., http://www.renesys.com/blog/2010/11/chinas-18-minute-mystery.shtml, 2010.

[13] Network Working Group. BGPSEC Protocol Specification. https://http://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol-04, July 2012.

[14] National Cyber Range Overview. http://www.dtic.mil/dtic/tr/fulltext/u2/a551864.pdf.

[15] DETER Testbed. http://deter-project.org/.

[16] CAIDA. AS Relationships Dataset. http://www.caida.org/data/active/as-relationships/, March 2009.

[17] RouteViews. RouteViews Dataset. http://www.routeviews.org/.

[18] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. Raptor: routing attacks on privacy in tor. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 271–286, 2015.

[19] Ripe ris raw data. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data.

[20] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Italo Cunha, Phillipa Gill, and Ethan Katz-Bassett. Investigating interdomain routing policies in the wild. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 71–77. ACM, 2015.

[21] Ying Zhang, Zhuoqing Morley Mao, and Jia Wang. Low-rate tcp-targeted dos attack disrupts internet routing. In *NDSS*. The Internet Society, 2007.

[22] Ahren Studer and Adrian Perrig. The coremelt attack. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS 2009)*, September 2009.

[23] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard A. Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In Ehab Al-Shaer, Somesh Jha,

and Angelos D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 635–647. ACM, 2009.

[24] Brent ByungHoon Kang, Eric Chan-Tin, Christopher P. Lee, James Tyra, Hun Jeong Kang, Chris Nunnery, Zachariah Wadler, Greg Sinclair, Nicholas Hopper, David Dagon, and Yongdae Kim. Towards complete node enumeration in a peer-to-peer botnet. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS*, pages 23–34. ACM, 2009.

[25] Kotikalapudi Sriram, Doug Montgomery, Oliver Borchert, Okhee Kim, and D. Richard Kuhn. Study of BGP peering session attacks and their impacts on routing performance. *IEEE Journal on Selected Areas in Communications*, 24(10):1901–1915, 2006.

[26] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet path performance. *SIGCOMM Comput. Commun. Rev.*, 36(4):375–386, 2006.

[27] Danny Dolev, Sugih Jamin, Osnat Mokryn, and Yuval Shavitt. Internet resiliency to attacks and failures under BGP policy routing. *Comput. Netw.*, 50(16):3183–3196, 2006.

[28] Network Working Group. RFC4724 — Graceful Restart Mechanism for BGP. `http://www.rfc-editor.org/rfc/rfc4724.txt`, January 2007.

[29] C. Villamizar, R. Chandra, and R. Govindan. BGP Route Flap Damping. RFC 2439 (Proposed Standard), November 1998.

[30] J. Cowie, A. Ogielski, B. Premore, and Y. Yuan. Global routing instabilities during Code Red II and Nimda worm propagation, 2001.

[31] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.

[32] Mohit Lad, Xiaoliang Zhao, Beichuan Zhang, Daniel Massey, and Lixia Zhang. Analysis of BGP update surge during Slammer worm attack. In Samir R. Das and

Sajal K. Das, editors, *IWDC*, volume 2918 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2003.

[33] S.M. Bellovin and E.R. Gansner. Using Link Cuts to Attack Internet Routing. 2004.

[34] H. Ballani, P. Francis, and X. Zhang. A study of prefix hijacking and interception in the Internet. *ACM SIGCOMM Computer Communication Review*, 37(4):276, 2007.

[35] K. Butler, T.R. Farley, P. McDaniel, and J. Rexford. A survey of bgp security issues and solutions. *Proceedings of the IEEE*, 98(1):100 –122, jan. 2010.

[36] Stephen T. Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure border gateway protocol (s-bgp) - real world performance and deployment issues. In *NDSS*, 2000.

[37] Tao Wan, Evangelos Kranakis, and Paul C. van Oorschot. Pretty secure bgp, psBGP. In *NDSS*. The Internet Society, 2005.

[38] Paul C. van Oorschot, Tao Wan, and Evangelos Kranakis. On interdomain routing security and pretty secure BGP (psBGP). *ACM Trans. Inf. Syst. Secur.*, 10(3), 2007.

[39] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the internet. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 149–160, New York, NY, USA, 2007. ACM.

[40] Greg Sinclair, Chris Nunnery, and Brent ByungHoon Kang. The waledac protocol: The how and why. In *In proceeding the 4th IEEE International Conference on Malicious and Unwanted Software (MALWARE)*, pages 69–77. IEEE Computer Society, October 2009.

[41] Maxmind. Geo IP to ASN dataset. http://geolite.maxmind.com, 2009. December 3, 2009.

[42] Network and Customer Installation Interfaces — Asymmetric Digital Subscriber Line (ADSL) Metallic Interface. Technical Report 2, American National Standards Institute, 1998.

[43] Q. Wu, Y. Liao, T. Wolf, and L. Gao. Benchmarking BGP routers. In *IEEE 10th International Symposium on Workload Characterization, 2007. IISWC 2007*, pages 79–88, 2007.

[44] Di-Fa Chang, Ramesh Govindan, and John Heidemann. An empirical study of router response to large BGP routing table load. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pages 203–208, New York, NY, USA, 2002. ACM.

[45] Scott A. Crosby and Dan S. Wallach. Denial of service via algorithmic complexity attacks. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, pages 3–3, Berkeley, CA, USA, 2003. USENIX Association.

[46] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. In *SIGMETRICS*, pages 307–317, 2000.

[47] Team Cymru. Team Cymru Bogon List. `http://www.team-cymru.org/Services/Bogons/bogon-dd.html`.

[48] Geoff Huston. BGP Routing Table Analysis Reports. `http://bgp.potaroo.net/`.

[49] A.C. Popescu, B.J. Premore, and T. Underwood. Anatomy of a leak: AS9121. Renesys Corp., `http://www.renesys.com/tech/presentations/pdf/renesys-nanog34.pdf`, 2005.

[50] Team Cymru. Team Cymru Secure BGP Template. `http://www.team-cymru.org/ReadingRoom/Templates/secure-bgp-template.html`.

[51] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480 (Informational), February 2012.

[52] Sharon Goldberg, Michael Schapira, Peter Hummon, and Jennifer Rexford. How secure are secure interdomain routing protocols. *SIGCOMM Comput. Commun. Rev.*, 41:87–98, August 2010.

[53] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David P. Mankins, and W. Timothy Strayer. Decoy routing: Toward unblockable

internet communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2011.

[54] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[55] Amir Houmansadr, Giang T.K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[56] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium*, pages 21–21. USENIX Association, 2004.

[57] JAP: The JAP anonymity & privacy homepage. `http://www.anon-online.de`.

[58] Ultrareach Internet Corporation. Ultrasurf - proxy-based internet privacy and security tools. `http://ultrasurf.us`.

[59] Knock knock knockin' on bridges' doors. `https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors`.

[60] New blocking activity from iran, Sep, 14, 2011. `https://blog.torproject.org/blog/iran-blocks-tor-tor-releases-same-day-fix`.

[61] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[62] CAIDA AS relationship dataset. `http://www.caida.org/data/active/as-relationships/index.xml`.

[63] Berkman Center for Internet & Society. Mapping local internet control. `http://cyber.law.harvard.edu/netmaps/geo_map_home.php`.

[64] E. Rosen and Y. Rekhter. BGP/MPLS IP Virtual Private Networks (VPNs). RFC 4364 (Proposed Standard), February 2006. Updated by RFCs 4577, 4684, 5462.

[65] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[66] Y. He, M. Faloutsos, and S. Krishnamurthy. Quantifying routing asymmetry in the internet at the as level. In *Global Telecommunications Conference, 2004*, volume 3 of *GLOBECOM '04*, pages 1474–1479. IEEE, 2004.

[67] Edward Wyatt. F.c.c., in a shift, backs fast lanes for web traffic. `http://www.nytimes.com/2014/04/24/technology/fcc-new-net-neutrality-rules.html?smid=pl-share&_r=0`. Accessed: 15 May 2014.

[68] Cisco visual networking index: Forecast and methodology, 2012 to 2017. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html`. Accessed: 15 May 2014.

[69] The NSA uses powerful toolbox in effort to spy on global networks. `http://www.spiegel.de/international/world/the-nsa-uses-powerful-toolbox-in-effort-to-spy-on-global-networks-a-940969.html`. Accessed: 15 May 2014.

[70] Bruce Schneier. Attacking tor: how the NSA targets users' online anonymity. `http://www.theguardian.com/world/2013/oct/04/tor-attacks-nsa-users-online-anonymity`. Accessed: 15 May 2014.

[71] Nicholas Weaver. A close look at the NSA's most powerful internet attack tool. `http://www.wired.com/2014/03/quantum/`. Accessed: 15 May 2014.

[72] Amir Houmansadr, Edmund L Wong, and Vitaly Shmatikov. No direction home: The true cost of routing around decoys. In *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium*, 2014.

[73] Joseph Menn. Secret contract tied nsa and security industry pioneer. http://www.reuters.com/article/2013/12/20/us-usa-security-rsa-idUSBRE9BJ1C220131220. Accessed: 16 February 2015.

[74] Phillipa Gill, Michael Schapira, and Sharon Goldberg. Let the market drive deployment: A strategy for transitioning to bgp security. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 14–25. ACM, 2011.

[75] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. *ACM SIGCOMM Computer Communication Review*, 41(4):75–86, 2011.

[76] World bank global indicators. http://data.worldbank.org/indicator/.

[77] Peeringdb. https://www.peeringdb.com/.

[78] Global internet phenomena report. https://www.sandvine.com/trends/global-internet-phenomena/.

[79] Netflix open connect program. https://openconnect.netflix.com/en/.

[80] Ip transit forecast. https://www.telegeography.com/research-services/ip-transit-forecast-service/index.html. Accessed: 26 August 2015.

[81] Brazil, europe plan undersea cable to skirt u.s. spying. http://www.reuters.com/article/2014/02/24/us-eu-brazil-idUSBREA1N0PL20140224. Accessed: 26 February 2014.

[82] Nick Feamster. An unprecedented look into utilization at internet interconnection points. https://freedom-to-tinker.com/blog/feamster/the-interconnection-measurement-project/. Accessed: 11 March 2016.

[83] Dark fiber community. http://www.darkfibercommunity.com/.

[84] Broadcasting Board of Governors. Broadcasting board of governors: Internet anti-censorship fact sheet. http://www.bbg.gov/wp-content/media/2015/04/Anti-Censorship-Fact-Sheet-10182015.pdf. Accessed: 28 Aug 2015.

[85] G. Huston and G. Michaelson. Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs). RFC 6483 (Informational), February 2012.

[86] S. Bellovin, R. Bush, and D. Ward. Security Requirements for BGP Path Validation. RFC 7353 (Informational), August 2014.

[87] Jeanne Sahadi. What the nsa costs taxpayers. `http://money.cnn.com/2013/06/07/news/economy/nsa-surveillance-cost/`. Accessed: 28 Aug 2015.

[88] Xueyang Xu, Z Morley Mao, and J Alex Halderman. Internet censorship in china: Where does the filtering occur? In *Passive and Active Measurement*, pages 133–142. Springer, 2011.

[89] Charles Reis, Steven D Gribble, Tadayoshi Kohno, and Nicholas C Weaver. Detecting in-flight page changes with web tripwires. *NSDI*, 8:31–44, 2008.

[90] Anonymous. The collateral damage of internet censorship by dns injection. *SIGCOMM Comput. Commun. Rev.*, 42(3):21–27, June 2012.

[91] Lennart Haagsma. Deep dive into quantum insert. `http://blog.fox-it.com/2015/04/20/deep-dive-into-quantum-insert/`. Accessed: 15 May 2015.

[92] Mukarram Bin Tariq, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. Detecting network neutrality violations with causal inference. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 289–300, New York, NY, USA, 2009. ACM.

[93] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, P Krishna Gummadi, Ratul Mahajan, and Stefan Saroiu. Glasnost: Enabling end users to detect traffic differentiation. In *NSDI*, pages 405–418, 2010.