DISSERTATION

# PROTECTING CRITICAL SERVICES FROM DDOS ATTACKS

Submitted by

Vamsi K. Kambhampati

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2012

Doctoral Committee:

Advisor: Daniel Massey
Co-Advisor: Christos Papadopoulos

Michelle M. Strout
Edwin K. P. Chong

ABSTRACT

PROTECTING CRITICAL SERVICES FROM DDOS ATTACKS

Critical services such as emergency response, industrial control systems, government and banking systems are increasing coming under threat from Distributed Denial of Service (DDoS) attacks. To protect such services, in this dissertation we propose Epiphany, an architecture that hides the service IP address making it hard for an attacker to find, attack and disable the service. Like other location hiding based approaches, Epiphany provides access to the service through numerous lightweight proxies, which present a very wide target for the attacker.

However, unlike these solutions Epiphany uses a novel approach to hide the service from both clients *and* proxies, thus eliminating the need to trust proxies or apply a filtering perimeter around the service destination. The approach uses dynamically generated hidden paths that are fully controlled by the service, so if a specific proxy misbehaves or is attacked, it can be promptly removed. Since the service cannot be targeted directly, the attacker may target the proxy infrastructure. To combat such threats, Epiphany separates the proxies into setup and data proxies. Setup proxies are only responsible for letting a client make initial contact with the service, while data proxies provide further access to the service. However, the setup proxies employ *IP anycast* to isolate the network into distinct regions. Connection requests generated in a region bounded by an anycast setup proxy are automatically directed to that proxy. This way, the attacker botnet becomes dispersed, i.e., the attacker cannot combine bots from different regions to target setup proxies in specific networks. By adding more anycast setup proxies, networks that only have legitimate clients can be freed from the perils of unclean networks (i.e., networks with attackers). Moreover, the attacker activity becomes more exposed in these unclean networks, upon which the operators may take further action such as re-

move them or block them until the problem is resolved. Epiphany data proxies are kept private; the service can assign different data proxies to distinct clients depending on how they are trusted. The attacker cannot disrupt on-going communication of a client who's data proxy it does not know. We evaluate the effectiveness of Epiphany defenses using simulations on an Internet scale topology, and two different implementations involving real Internet routers and an overlay on PlanetLab.

# ACKNOWLEDGEMENTS

It is my pleasure to thank everyone who supported me with my PhD at Colorado State University. In particular, the strong support the computer science department community and my family which helped me accomplished this task. I must specifically thank my advisors Dr. Dan Massey and Dr. Christos Papadopoulos for their valuable advise and support in achieving my PhD. Not only did I learn a great deal from them, but they also enlightened me to the ancient Indian concept of "*Guru Devo Bhava*" (Teacher is God). As my teachers, they guided and motivated me in numerous ways. I understand now that persistence is an important part of the PhD process. I would also like to thank Dr. Michelle Strout and Dr. Edwin Chang for serving on my committee, and keeping on open eye (and ear) when I needed outside opinion. I am indebted to my supervisors at Nokia: Dr. Lars Eggert and Dr. Pasi Sarolahti for giving me a great opportunity to work with the EU partnered Trilogy project. It is from the exposure I got at this project, that I came upon the main idea behind Epiphany. I would also like to thank my colleagues Yan He, Steve DiBenedetto, Kaustubh Gadkari and my friends who were constantly there for open discussion. They offered me plenty of constructive criticism. This project would not have shaped the way it is without their input. I must thank my parents and my wife for their unconditional support and specifically keeping me motivated through the lows. Finally, I would like to thank the anonymous reviewers for their insightful comments, which greatly helped improve the clarity of this work.

TABLE OF CONTENTS

LIST OF ACRONYMS

**AS** Autonomous System

**DoS** Denial of Service

**DDoS** Distributed Denial of Service

**DoC** Denial of Capabilities

**DNS** Domain Name System

**DP** Data Proxy

**SP** Setup Proxy

**SOS** Secure Overlay Services

**SSM** Single Source Multicast

**TOR** The Onion Router

**TVA** Traffic Validation Architecture

**rps** Requests Per Second

**dps** Data Packets Per Second

**RTT** Round Trip Time

**TTL** time-to-live

LIST OF FIGURES

# Chapter 1

# Introduction

As critical systems such as emergency response, industrial control systems, government and banking systems are moving to the public Internet, they become susceptible to Distributed Denial of Service (DDoS) attacks; examples include the 2007, 2008 attacks on government websites [27, 28] and the 2010 attacks on financial institutions [20]. A recent survey [29] also concludes that these attacks are the top-most threat to Internet services. In DDoS, an adversary employs several hosts (bots) spread across in the Internet to send large volume of traffic to a target destination in order to consume its finite resources (bandwidth, CPU). As a result, the attacker prevents a number of legitimate clients from using the service. Damages range from loss of revenue, data loss, to defamation and in some extreme cases danger to lives. Critical services are different from regular Internet services, they can tolerate lower bandwidths, may be willing to spend more on resources, and assume community support from Internet service providers and users. However, by definition they must have high availability, which becomes extremely challenging when DDoS attacks can reach 100 Gbps [29] (up from 400Mbps in 2002). Solutions such as firewalls are ineffective under such loads, and this is often true of commercial solutions [1, 39], which also tend to be expensive. The research community has many proposals to defend against DDoS attacks [8, 19, 21, 34, 36, 46–48], but most are not aimed at critical services and to our knowledge none has been widely deployed in the Internet.

Critical services are often treated as a separate class of applications, assumed to be small in number and low in bandwidth, with proposed defenses based on *location hiding* principles [5,12,17,38]. The core theme around these approaches is to effectively

hide the service address so that attackers cannot target it directly. Typically the service becomes accessible through a large number of *proxies*, which tend to dilute the attack. However, given enough firepower an attacker can still overwhelm the proxy network or portions of it to mount an effective attack.

In this dissertation, we develop a novel DDoS defense architecture called *Epiphany*. Unlike previous approaches, Epiphany eliminates the need for a service network address making it virtually impossible to attack the service directly. This is in contrast to approaches that while hiding the service location from the general public, still expose the service address to a small set of trusted proxies. Such proxies could become a primary target for the attacker, or worse, if compromised would expose the service address. For instance, TOR [12, 30, 31] and SOS [17] have a small set of proxies that know the service address (guard nodes in TOR [30] and secret servlets in SOS [17]), but use multiple layers of indirection to keep them secret. While Epiphany still relies on proxies, these are untrusted and cannot reach the service without the service allowing it. Proxies send traffic over ephemeral *hidden paths*, which are created on the underlying network using a technique akin to multicast routing. The paths are under complete control of the service and can be easily removed if a proxy misbehaves.

Proxies can still be discovered and attacked, so Epiphany uses *anycast* to limit proxy access. Since anycast packets are directed to the local proxy by the underlying routing system, an attacker is forced to have bots in the same anycast region as the clients to attack their proxy. However, since Epiphany proxies are lightweight (explained later), many can be quickly deployed and the anycast regions can be made smaller (down to subnet level if needed) in an effort to isolate attackers. This dilution can not only help legitimate clients circumvent attackers, but can make it easier to expose bots whose activity becomes more visible.

Another feature of Epiphany is the separation of setup proxies (SP) from data proxies

(DP). Epiphany makes the SPs public, but keeps the DPs private so they are harder to attack. A client must first request access from the service using an SP, but the service will decide if it should authorize the client. Unlike TOR [12] where the client selects its DP, an Epiphany DP is revealed to a new client by the service, perhaps after gaining trust. The service further divides DPs to different levels of trust, assigning clients accordingly. This further isolates new, yet-untrusted clients from existing trusted ones. The service also has full control over the location of DPs, which enables physical separation based on trust to avoid collateral damage.

Finally, in order for clients to learn about proxies, a discovery service in necessary. But this service is often a target for the attacker. Epiphany uses the DNS for proxy discovery, which inherits all the benefits (and disadvantages) of DNS. Typical measures to make DNS robust apply here: multiple servers, long time-to-live (TTL) values, robust network and hardware and so forth.

In summary, using location hiding Epiphany makes it difficult for an attacker to pinpoint and target a protected critical service: the attacker cannot attack the service directly since there is no network address to target. Moreover, the attacker is sandboxed by anycast and has to go through a trust-building process before it can attack a service. Even so, the attack is localized to a private proxy, which can be easily and quickly disconnected by the service, and will only affect clients currently using that proxy.

## 1.1 Thesis Statement

"Our thesis is that hiding the service destination address and providing access to the service through a large front-end of proxies that themselves do not know the destination address, and that form distinct regions in network, and that are issued differently to distinct clients based on trust can effectively defend against large scale Distributed

Denial of Service attacks. Specifically, we make the service resilient to inside attacks from compromised proxies by hiding the destination address from them and providing them with a temporary path that can be removed at any time to prevent them from causing harm. We isolate and localize attackers to smaller network regions so that they are dispersed, become easier to defend against or can be discovered and removed. We allow the destination to issue different proxies to different clients such that an attacker cannot easily harm the data exchange between the destination and a legitimate client. Finally, we make proxy discovery robust by making the clients learn about a small number of proxy addresses that remain valid for a long duration, but are associated with a large group of ever changing proxies."

## 1.2 Contributions

In this dissertation, we develop a strong location hiding architecture where not only is the destination hidden from the sources but even the proxies do not know its address. We show that this architecture is a viable DDoS defense solution through simulations and experiments on two testbeds, one running on real Cisco routers and another as an overlay on PlanetLab hosts. The specific contributions of our work are as follows:

1. Epiphany achieves strong location hiding property in which neither the proxies nor the sources know about the destination address. Epiphany does not place trust on the proxies or requires trusted components to operate. In order for a proxy to reach the destination, we develop a novel *hidden paths* mechanism. Hidden paths share many similarities with multicast routing, and can be implemented on routers without sacrificing efficiency when forwarding traffic. Also, most of the multicast functionality available on routers could be reused to implement hidden paths. They also do not require hardware or architectural changes to routers.

2. Although IP anycast has been used to protect DNS servers from DDoS attacks,

to our knowledge no other location hiding architecture has applied IP anycast to render attackers powerless. Using IP anycast in Epiphany has two benefits; first the anycast SPs separate the attackers and dilute them (i.e., attackers can no longer muster strength with numbers) and second, anycast SPs have a very small address footprint so proxy discovery becomes simpler and can be made robust. In addition, anycast SPs localize attackers to smaller regions, which could lead to the discovery of the bots, or they can be handled using techniques such as client puzzles [43].

3. We provide a model in which the service can assign different data proxies to client depending on how a client is trusted. This model is different from making all proxies public and letting the client chose a proxy. We believe that making the proxies public will allow the attacker to strategize and disable portions of the proxies at a time to disrupt some legitimate clients. If an important legitimate client happens to be on a targeted proxy, its communication will be disrupted. Our approach, on the other hand, has the potential to prevents these situations. The attacker does not know all data proxies and hence is limited to targeting setup proxies.

4. We construct a small scale Epiphany testbed using Cisco routers and prove that; 1) It is possible to completely hide the destination network by disabling route announcements for the destination network into the global Internet. 2) Hidden paths can be constructed out of multicast routing functions available on current routers. Specifically, Single Source Multicast (SSM) can be used for this purpose. 3) Hidden paths can be quickly added and deleted in the order of milliseconds to seconds respectively. Thus, new proxies could be easily added and compromised proxies can be removed quickly once they are identified.

5. We conduct anycast SP simulations on an Internet scale topology and show that; 1) for a small legitimate client density of 0.01%, only a 0.5% SP density is sufficient to suppress over 93% of the attackers even when 10% of the entire network is occupied by attackers. 2) Anycast SPs effectively localize attackers to within 3 to 4 hops of the SP, and most often there is only one attacker interfering with a legitimate client at an SP. 3) Placing the SPs closer to the edge networks, i.e., close to where the attacker and legitimate clients are located is the best strategy for SP placement, while placing them so that there is at least one SP per AS is a better strategy when there are far fewer SPs.

6. We implemented the various components of Epiphany as an overlay running on PlanetLab [3] and prove that separating the proxies helps limit collateral damage to legitimate clients. As a comparison, we implemented a proxy that performs both setup and data transfer functions and show that data transfer time increases dramatically with the attacker's sending rate, while in the separated proxies case, the data transfer time is independent of the attacker's sending rate.

## 1.3 Organization

The rest of this thesis is organized as follows. In Chapter 2 we present background on DDoS attacks and describe past defenses that are location hiding based and non-location hiding based. We then develop a taxonomy of network capability architectures, which are a new but an important approach in DDoS defense in Chapter 3. The taxonomy helped us identify several new design directions. We apply some of these lessons to develop the Epiphany location hiding architecture. The Epiphany architecture itself is discussed in Chapter 4. The main components of this architecture are setup proxies, data proxies, and hidden paths. Epiphany has many parallels with capability architectures. For instance, hidden paths in Epiphany can be thought as non-cryptographic capabilities;

without them, proxies cannot send packets to the destination. The destination pushes hidden paths to proxies, and the hidden paths can be explicitly town down (revoked) at any time by the destination. We evaluate the effectiveness of Epiphany defenses in Chapter 5. Finally, we conclude in Chapter 6 with a discussion of how to deploy Epiphany in the Internet, and describe our future directions.

# Chapter 2

# Background & Related Work

In this chapter we discuss the DDoS problem in more detail and explain why it is a difficult problem to solve in the Internet. We then summarize several prior defenses starting with the ones most similar to Epiphany, i.e., location hiding based approaches, and then discuss non location hiding based approaches.

## 2.1 DDoS Attacks

The Internet communication model is simple; a client that has requests/data to send learns the receiver's address, wraps the request/data in a packet, stamps the receiver's address on the packet and delivers it to the underlying network. The network carries the packet through intermediate routers and eventually delivers it to the receiver. If the network is unable to deliver the packet, it simply drops the packet. The ends are responsible for detecting failures and handling them. The ends engage in a communication protocol to inform each other to either slow down, retransmit or speed up the packet transfer depending on how fast they can process packets and how fast the network is able to deliver the packets. Unfortunately, this model of communication has a fundamental problem; the receiver has no control over the sender. The receiver can ask the sender to slow down if it is unable to process packets fast enough, but the sender could be a broken piece of software that does not comply to these requests. The sender may also be a malicious adversary that is intensionally overloading the receiver. Such is the case in DDoS attacks.

In these attacks, an attacker deliberately prevents legitimate clients from accessing a service by targeting the service host through a distributed set of nodes. The attacker

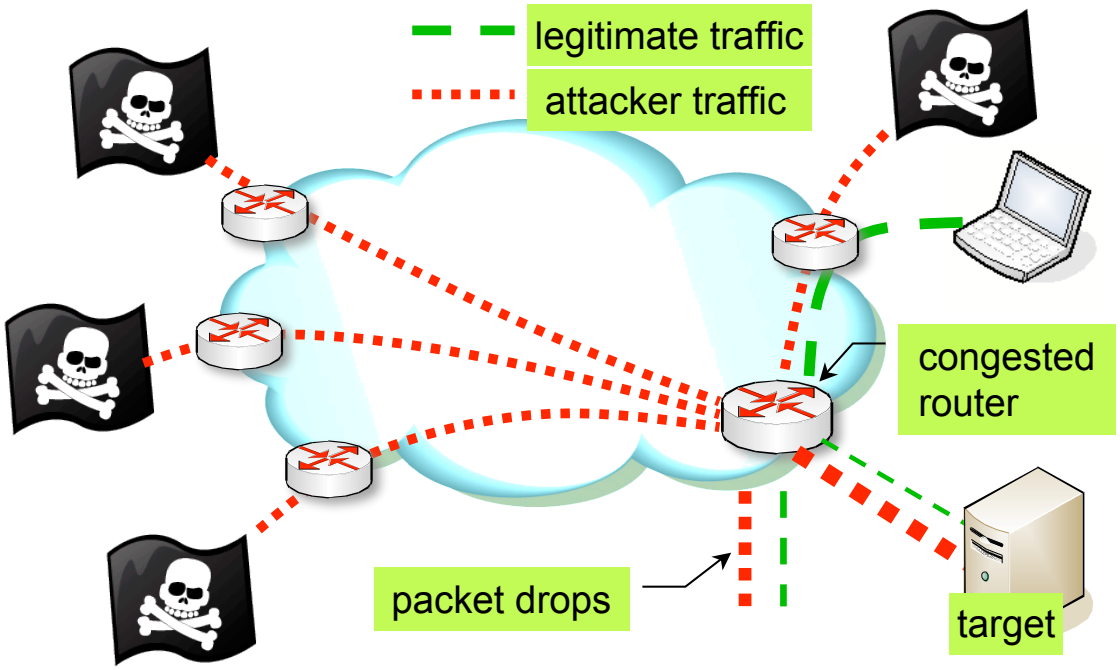


**Figure 2.1.1:** An example of a DDoS flooding attack

may consume important resources on the paths to the service, such as network bandwidth, router CPU or memory, may attack the infrastructure that the service relies upon, or target the service directly and consume its resources (CPU, memory, bandwidth) to render the service unusable to legitimate clients. Figure 2.1.1 shows an example of *bandwidth exhaustion* attack, also known as flooding attack. The attacker employs several distributed hosts to send large volume of traffic to the target, which becomes aggregated as it reaches the destination and causes severe congestion on some routers. At the same time, a legitimate client may be attempting to reach the service or may be already communicating with the service. Although the congested router will drop the attacker traffic, without further knowledge into what flows are important/legitimate, the router indiscriminately drops the legitimate traffic as well. The packet loss causes serious degradation or complete disruption of the service to the legitimate client.

## 2.2 Why DDoS is a Hard Problem

From the above description, one can observe that the underlying cause of DDoS is perhaps due to the lack of *distributed enforcement*. Making the ends responsible for dealing with all decisions regarding the communication presents the attacker with an opportunity to misbehave. The destination may set forth any protocol rules for a sender to follow, but the attacker will simply ignore the rules. One might argue that the network should participate in enforcing the rules set forth by the destination, but there are several problems associated with that approach.

First, network providers may be reluctant to modify the basic service model since it lacks strong incentives for the operators and moreover distributed enforcement increases overhead on the routers. The operator's goal is to try to optimize packet deliver and but they care less about what is being delivered. Although this attitude may be changing in recent years due to the increasing number of DDoS attacks [29], distributed enforcement requires cooperation among the different parties involved, which may be difficult to achieve in the Internet.

Second, identifying bad traffic is a difficult problem on its own. Given a packet, the network cannot easily identify if it is originating from an attacker or from a legitimate client. There are instances in which the service might appear under attack, but in reality, there may be a sudden surge in demand causing a large number of legitimate clients to flock to the service and rendering it unusable (these cases are referred as *flash crowds*, or *slashdot effect*). Moreover, the attackers may spoof their IP addresses and masquerade as legitimate clients, i.e., the addresses used in the source IP field of packets sent by the attacker may contain addresses of legitimate hosts. Address spoofing complicates traffic classification and enforcement in the network, and if not done correctly, a legitimate client could be blamed that has nothing to do with the attack.

## 2.3 DDoS Defenses

To remedy the DDoS problem, many solutions have been proposed in the literature [5, 6, 8, 12, 15, 17–19, 21, 22, 30, 31, 34, 36, 38, 42, 45–48], but to the best of our knowledge, no one solution has been widely deployed in the Internet, and moreover most solutions are not designed with critical services in mind. We structure this discussion on DDoS defenses around the main topic of this dissertation, i.e., location hiding, and group the solutions as location hiding based or non location hiding based.

### 2.3.1 Location Hiding Architectures

Epiphany belongs in the general class of location hiding architectures [5, 12, 17, 18, 30, 31, 38], but is more similar to TOR hidden services [12, 30, 31] at an abstract level. These architectures hide the destination address from the general public and make the service accessible only through proxies. Since the attacker cannot reach the destination directly, he is forced to target the proxies instead. Location hiding architectures employ large number of proxies to present a very wide target for the attacker, and also frequently vary the active proxy set to evade the attacker.

TOR [12] provides location hidden services through an overlay based anonymization network, while Epiphany uses the underlying network to hide the service and is not designed to provide anonymity. In TOR, a hidden destination selects a set of TOR overlay nodes called *introduction points* and constructs secure paths to these points using the TOR overlay nodes. The destination announces the service name and the introduction points to a TOR directory service. A source that wishes to receive service contacts the directory service and retrieves the list of introduction points. It then picks one of the introduction points and uses another TOR node called *rendezvous point* to establish connection with the destination. Note that both the source and destination do not know about each other's IP addresses. After this process, they exchange traffic via the

rendezvous point.

Conceptually, TOR introduction points and rendezvous points can be viewed as setup and data proxies of Epiphany. However, an important distinction is that Epiphany lets the service decide which DPs to assign to a client whereas TOR lets the client choose the rendezvous point. Letting the client choose the DP can be disadvantageous in Epiphany, because an attacker could exploit this and make the destination build hidden paths to colluding DPs and make the destination expend resources in removing those DPs.

TOR secure paths are functionally similar to Epiphany hidden paths, but they use strong cryptographic mechanisms to encrypt all traffic, including control traffic for constructing hidden paths. Each intermediate hop in the TOR secure path only knows about its adjacent neighbors, similar to the routers in Epiphany hidden paths, which only know about the adjacent routers on the path. However, since the secure paths in TOR are constructed on top of end-user hosts, any node in the TOR overlay could become the predecessor of a hidden server. Overlier et al [30] show that an attacker could game the system and become this first hop node. Once there, the attacker could use timing analysis to discover the hidden destination.

Epiphany does not suffer from this type of attack, since the hidden paths in Epiphany always start at the border routers in the destination network. The attacker cannot arrange for its nodes to become these border routers. [30] suggests using a set of trusted/preferred overlay nodes called *entry guard nodes* to prevent the attacker from becoming one of the first hop nodes. However, the attacker may discover the entry guard nodes and target them instead. To combat this, the possible entry guard set has to be large, but making the set large and also ensuring that they are trust-worthy is a challenging problem. The border routers in Epiphany hidden network are trusted, but the attacker would need to learn about them to target the service. However, since the service location is hidden, the

security of our system depends on how the operators keep this information secret.

TOR introduction points are vulnerable to DDoS attacks much the same way SPs in Epiphany are vulnerable to attacks. To protect the introduction points, *Valet Service* [31, 32] proposes to hide them behind another layer of indirection involving large number of *valet nodes*. In this approach, a client learns about a *contact information ticket* that has information about a valet node and an encrypted valet token. The valet token contains the address of the introduction point, which only the valet node can decrypt. To reach the service, the client contacts the valet node and asks it to forward the request to the introduction point listed in the valet token. The valet node contacts the introduction point on behalf of the client and the connection process completes as before (i.e., the introduction point tells the service about the client's rendezvous point). Valet service gives different contact information tickets to different clients based on the information/trust regarding a client. Attackers will need to learn about all possible contact information tickets to learn about all valet nodes, and even then, they will not know about the introduction points.

This model is very different from Epiphany's setup model. Epiphany does not hide the service name, nor does it make an attempt to hide the setup proxy addresses. But rather Epiphany uses anycast to scale the SPs behind a single address and uses additional defenses such as client puzzles to protect the connection setup process of legitimate clients. Unlike valet services, which separates clients using different contact information tokens, Epiphany separates clients at data level, using different data proxies.

The original TOR approach uses a central directory service to allow clients to learn about the proxies, but the directory service is vulnerable to DDoS attacks. [31] proposes to address this issue using a decentralized directory service. Unlike this approach, Epiphany uses DNS to provide proxy discovery, but makes the discovery service resilient to attacks using known techniques for making DNS robust.

Secure Overlay Services (SOS) [17] and its generalized cousin Mayday [5] build an overlay of proxies and designate different proxies to hold different information about the overlay and information about the destination. All proxies are publicly accessible, but authenticate clients before letting their traffic enter the proxy network. Once the traffic is in, the overlay routes it to a set of proxies called *beacons* that are responsible for forwarding the traffic to yet another set of proxies called *secret servlets*. Only the beacons know about the secret servlets, and only the servlets know about the destination's address. The servlets are ultimately responsible for forwarding the traffic to the destination. SOS establishes a filtering perimeter around the destination and only permits packets from these secret servlets. At any time, the attacker does not know the identities of the servlets, and hence cannot forge packets with their IP addresses to pass through the filtering perimeter. In addition, SOS changes the set of proxies acting as beacons and secret servlets periodically, so that even if an attacker catches up and learns about the servlets or targets them, the service is not rendered unavailable. Portions of the overlay may be affected due to an ongoing attack, but the overlay recovers and works around the failures.

Unlike SOS, Epiphany does not authenticate its clients at the proxies. We believe distributing authentication information to all proxies presents a different set of challenges. Epiphany proxies are lightweight, i.e., they do not hold user authentication information and hence they are more deployable. Moreover, Epiphany does not require a filtering perimeter since it never reveals the destination's IP address to anyone in the network. Finally, Epiphany separates its proxies so that attacks on its SPs do not affect clients talking to the service via the DPs. In other words, Epiphany reduces collateral damage to a legitimate client since the attacker does not know which DP a client may be using to access the service. If an attacker targets some proxies in SOS, clients using those proxies will be affected. However, SOS has one benefit over Epiphany, an SOS

client can freely move to a different proxy if its lost to an attack. Proxy failures (such as an SP failing) may cause brief interruptions to clients in Epiphany.

Internet Indirection Infrastructure ($i3$) [18,38] constructs an overlay on top of the Internet and changes the current Internet model of associating an IP address (an identifier) and with a host location. Hosts in $i3$ do not know about the IP addresses of each other, but rather send packets to logical *identifiers*. A receiver specifies interest to receive packets through these identifiers. A service that wishes to be globally visible advertises a few public identifiers. The client learns about these identifiers and contacts the service by sending a request to the identifier. The underlying $i3$ overlay delivers the request to the actual destination, which may then gives the client a different set of private identifiers for rest of the communication. An attacker in $i3$ could flood the public identifiers to flood the service, but $i3$ limits such attacks using proof of work based on computational puzzles [18]. Although $i3$ hides the IP addresses, an attacker may find out about the host IP address of an $i3$ server and target it directly through the Internet.

In general, overlay based location hiding approaches such as SOS, $i3$ and TOR must deal with penetration attacks in which the attacker progressively compromises a series of nodes in the overlay to identify the destination. Ju Wang *et al* [41] study penetration attacks and point out that reconfiguration, proxy depth and host diversity are important factors in combating these attacks. Unlike overlay based approaches, Epiphany constructs its hidden paths on network routers, and hence is only as strong as the security of these devices on the underlying network. Compromise of these devices would have implications far beyond preventing access to a single service.

Network authenticated path panning (SNAPP) [33] describes a mechanism by which hidden paths could be constructed using less state at routers. In this approach, routers encrypt the upstream and downstream path information and encode that information in the packets to aid packet forwarding. However, this approach requires changing the IP

packet structure and forwarding semantics at routers. While SNAPP can provide sender anonymity, its main purpose is to override routing protocol decisions.

### 2.3.2 Non-Location Hiding Based Defenses

Epiphany also leverages some ideas from non-location hiding based approaches. Content distribution networks (CDNs) such as Akamai CDN [1] present a very wide target for the attacker using numerous distributed replicated instances of the service to handle large traffic volumes. In a way, they engage in an arms race against the attacker by outmatching the attacker's ability to consume those resources. However, replication at such levels imposes a great amount of management overhead, and moreover, not all services can be easily replicated. For instance, services that are geographically tied to a location, databases holding sensitive information and so forth. Replicated instances are similar to proxies, but are not lightweight like Epiphany.

Phalanx [13] employs a large swarm of proxies called *mailboxes* to aid communication between the sources and a destination. The source is not allowed to send packets directly to the destination and any such attempts will be dropped at a filtering perimeter around the destination. Instead, the source engages in a setup process with the destination to retrieve a cryptographically generated sequence of mailboxes and nonces it can use to reach the destination. The source stamps each packet with the nonce and sends it the next mailbox in the sequence. The destination also knows about the next mailbox and the nonce and retrieves the next packet from the mailbox. Note that the destination picks up packets from the mailboxes at its own pace to avoid getting overwhelmed. Similar to $i3$, the setup process in Phalanx is protected using computational puzzles. Mailboxes do not have direct access to the destination either. The destination punches a temporary hole in the filtering perimeter as it retrieves a packet from a mailbox and immediately closes it so that a compromised mailbox cannot flood the destination.

Network capabilities [6, 34, 42, 46, 48] propose to fundamentally change the Internet

communication model. They require the sender to obtain cryptographically verifiable tokens called capabilities from a receiver as proof of authorization to send traffic to the receiver. The network actively participates in verifying the capabilities and treats packets with capabilities preferentially over packets without capabilities. Using capabilities, the receiver has explicit control over how much traffic the sender is allowed to send, and who is allowed to send. Filtering bad traffic is effective, since an unapproved source will not have capabilities and its traffic will be dropped in the network. Like Epiphany, capabilities have an open model, in which a source sends requests to obtain authorization/capability from the destination. However, the attacker may flood the request channel to prevent clients from receiving capabilities (Denial of Capabilities attacks [7]). We describe capabilities architectures in much more detail in Chapter [**?**].

In addition to the above defenses, there are other DDoS defenses based on filtering. In IP traceback [22, 36, 47] intermediate routers probabilistically select packets to mark with their IP addresses so that the destination can reconstruct the paths taken by offending traffic. The goal is that once the paths are identified, traffic filters could be installed deeper in the network to remove offending traffic that may arrive on those paths in the future. However, the attacker may spoof the IP address of the traffic it sends and make it difficult to generate filtering rules to remove such traffic. Path Identifier (Pi) [45] proposes to make traffic originating from the same network paths identifiable using markings in the packets, but does not attempt to reconstruct the path. A destination can request its upstream routers to filter traffic based on these markings rather than the source IP address.

In Pushback [15, 21] a congested router classifies incoming traffic to identify high bandwidth aggregates that are causing the congestion, and rate limit the aggregates to relive congestion on the bottleneck link. Subsequently, the congested router requests its upstream routers sending the aggregate to rate limit the traffic as well. These routers in

turn will push back the rate limiting request upstream until it reaches the source of the aggregate or close to the source of the aggregate. However, pushback may incorrectly drop legitimate traffic that is mixed with the aggregate traffic, and moreover, DDoS attackers may be highly distributed making it difficult to identify the high aggregate causing network paths.

AITF [8], and StopIt [19] propose to filter offending traffic at its source network since that is the most effective location in removing bad traffic early in the network. In AITF, the border routers in each autonomous system on the path to the destination add their IP addresses to packets when forwarding them. The destination uses these addresses to identify the originating network even if the source spoofs its IP address. StopIt eliminates source address spoofing using a path identification mechanism [19]. In both these architectures, when the destination identifies offending traffic it requests an agent in the source network to perform filtering. The agent in the source network confirms if indeed a source in its own network is sending the offending traffic and filters such traffic. However, to be effective, these solutions require cooperation from the source networks and require large scale deployment; otherwise any openings in the defensive perimeter will let unwanted traffic to reach the destination. Moreover, they must deal with fluctuating/varying attacks in which the attacker attempts to evade detection by varying the amount of traffic, or the attacker may resume the attack once the filtering request at the source network has expired.

# Chapter 3

# Taxonomy of Capability Architectures

Capability architectures [6, 26, 34, 42, 46, 48] changed the way we perceive DDoS defenses in a fundamental way. Rather than remove offending traffic after the fact, capability architectures give explicit control to the receiver in deciding and expressing what traffic it *wants* to receive. They place controls throughout the Internet to treat wanted traffic with higher priority. Traffic that is not deemed important may be rate-limited or dropped entirely. However, despite the many point solutions, capability architectures have not been studied rigorously to understand the tradeoffs and challenges. In this chapter we analyze capability architectures from ground up and develop a taxonomy to categorize existing approaches and identify new directions. We start with a brief description of the main categories of the taxonomy and describe how we arrived at them. We then describe the categories in greater detail and evaluate several capability implementations using the taxonomy framework. Finally we discussion a few open areas we identified through the taxonomy and motivate the need for a location hiding based approach to protect critical services.

## 3.1 Overview

Network capability architectures [34, 42, 46, 48] propose fundamental changes to the Internet service model, in that they require routers to treat packets with authorizations (i.e., capabilities) preferentially over other traffic. In these architectures, a sender that wants to communicate with a receiver needs to obtain an authorization from the receiver. To get this authorization, the sender starts out by sending an initial capability request packet to the receiver. Routers along the network path insert cryptographically generated

tokens into the request. The receiver gets the request and decides if it wants to grant authorization to the sender; if so, it cryptographically synthesizes a capability from the router tokens and returns it to the sender. Otherwise, the receiver ignores the request. The sender includes the capability in subsequent data, until either the capability expires, or is revoked by the receiver. The cryptographic operations on capabilities allow routers to easily verify the authorization and give preferential treatment to such traffic.

Capabilities allow the receiver to exert fine grain control over the senders. Senders have to be vetted initially, and misbehaving senders can be filtered by letting their capabilities expire. Filtering bad senders is efficient in terms of keeping unwanted traffic off the network, because in most cases routers near the source are likely to drop unwanted packets. In a broad sense, capability architectures take the receiver's choices to *receive*, *regulate*, or *reject* certain traffic and ask the network to enforce such choices.

However, despite the many point solutions [34,42,46,48] in capability architectures, there has not been a rigorous study of the entire solution space. Such a study is important since capabilities is a new idea and adding capabilities to the Internet introduces unexplored yet fundamental challenges and trade-offs due to the new service model; the implications of which are not well understood. To understand these challenges and tradeoffs, we take a fresh look at proposed capability architectures and develop a taxonomy. We believe that a taxonomy is necessary to expand the current understanding of the problem and solution space in capability architectures and architectures that are similar in essence to capabilities (for instance, $i3$ [18,38], SOS [17], Phalanx [13] are all authorization based architectures similar to capabilities). The taxonomy also provides a framework to compare various existing and future implementations of these type of architectures. Capability architectures have significant impact on the Internet in terms of, assimilating the new service model, changes to router hardware and end-host/router software. Factor such as these and incremental deployment strategies must be carefully

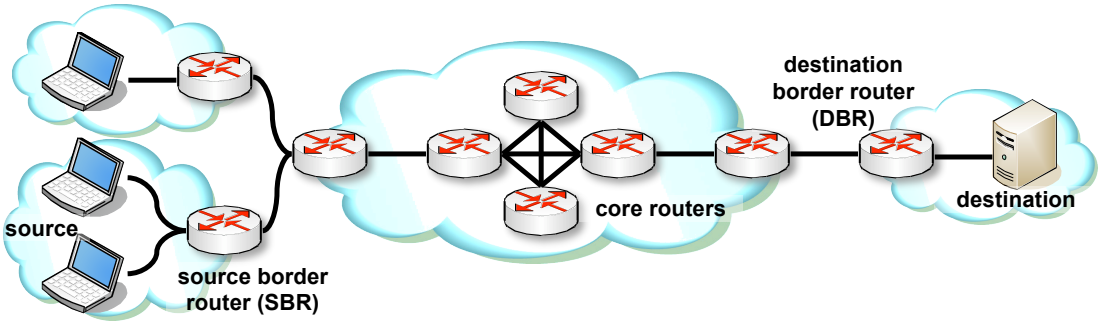


**Figure 3.1.1:** Network model of capability architectures

studied. In developing the taxonomy and studying the solution space, we have come to the following realizations:

- Capabilities alone are insufficient to remedy the DDoS problem as they are susceptible to colluding attacker problem. The colluding attacker problem is studied in [19, 48]. We expand on this in Section .
- The general capabilities approach is richer than what the existing approaches suggest. We identifier some new areas to explore, such as explicit revocation of authorizations, pushing capabilities to a sender, marking/deciding at locations other than the source/destination and non-cryptographic capabilities.

### 3.1.1 Network Model

To develop the taxonomy, we introduce a simple network model to identify various players involved in capability architectures. Our model omits intricacies of a physical network, but is specific enough to capture critical components of the network needed to describe the taxonomy. Figure 3.1.1 shows the network model.

The network has several sources, some of which might be DDoS attackers, and some are legitimate clients. We assume the legitimate clients are attempting to reach the destination, while the attackers are targeting it. The source(s) and destination are connected via routers, which could be malicious themselves. The network is distributed, i.e., dif-

ferent administrative authorities control different parts. For instance, the source and destination may be controlled by users, their respective networks controlled by organizations (companies, universities, etc), and the core network is controlled by Internet service providers (ISPs). The parties involved in distributed networks may not agree to the same policies, and may be managed with different levels of expertise. These properties complicate the deployment choices for capability architectures.

Our model identifies five important locations. The *source* and *destination*, as described above. These are connected to a *source border router* (SBR) and a *destination border router* (DBR) respectively. Without loss of generality, we assume packets from the source and destination pass through these routers (note that a site may have multiple border routers as with site-multihoming, but we ignore these cases to keep the model simple). We group the rest of the network into *core routers*. The policies at the core and border routers are likely different from the policies at the hosts.

### 3.1.2 Towards a Taxonomy

To defend DDoS attacks, capability architectures need to perform three important tasks. First, they need to provide authorizations to packets. We refer to this task as *traffic classification* in the taxonomy. Specifically, the architecture must provide mechanisms to decide what packets are authorized and then mark those packets. Second, they need to treat packets with authorizations preferentially at routers over packets without authorizations (or that have invalid authorizations). For instance, when a router is about to run out of resources, it may drop packets without authorizations, but allow authorized traffic to pass. We refer to this task as *enforcement*. Third, capability architectures need to coordinate the classification and enforcement processes and, to accommodate changes in decision and sender behavior they must ensure capabilities expire and are refreshed periodically. We refer to these tasks as *management*.

These three tasks form the top level categories in the taxonomy, namely. 1) Traffic

**Table 3.1:** Definitions used in the taxonomy

| Term | Definition |
|---|---|
| Wanted | Traffic with valid capabilities |
| Unwanted | Traffic with invalid capabilities |
| Unclassified | Traffic without capabilities |
| Decision | The process of determining whether a flow is wanted or unwanted. |
| Marking | The process of adding capability bits (given by a decider) to a packet. |
| Enforcement | The process of determining what forwarding action to take depending on the capability in a packet. |
| Setup | The process of initializing capabilities in the network for a specific source and destination. |
| Refresh | The process of keeping the capabilities active in the network. |
| Revocation | Explicitly removing the capabilities issued by a decider from the network. |

Classification – defines the actions relating to *decision* and *marking* packets into different types. 2) Enforcement – describes the process of taking necessary forwarding actions on packets, and 3) Capability Management – deals with *setup* and *refreshing capabilities* in the network. The precise definitions of the terms we use in the taxonomy are described in Table 3.1 and Figure 3.1.2 shows the resulting taxonomy.

The sub-categories in the taxonomy arise from fundamental questions such as, *what*, *where*, *how* and *when* an action is performed. For instance, the decision process involved in traffic classification requires understanding "what are the traffic classes?", "where are the decisions made?", "how to decide if a sender is authorized?", and "when to decide if authorizations are needed?". We use various metrics to argue about and understand the tradeoffs an architecture designer has to make when coming up with a new capability architecture implementation, or an architecture that is in essence similar to capabilities. The metrics include correctness, effectiveness, incentives, deployment issues, and effect on network resources (such as bandwidth, cpu and memory). For example, when choosing the location where decisions are made to grant capabilities, we use correctness of classification as the metric. In this case, the destination has the most knowledge

about the resources under attack and hence decisions made at the destination are more accurate.

## 3.2 A Taxonomy of Capability Architectures

We now discuss the taxonomy in more detail.

### 3.2.1 Traffic Classification: Decision

The first part of traffic classification process is to determine whether a flow is wanted or unwanted. We refer to the node making such decisions as the *decider*. Figure 3.1.2 shows the taxonomy of the decision process under traffic classification.

#### 3.2.1.1 Traffic Types

Capability architectures roughly divide packets into three types, those that have valid capabilities, i.e., *wanted* traffic, those that have invalid capabilities, i.e., *unwanted* traffic, and those that have no capabilities, or *unclassified* traffic. The first two categories follow from the fact that decisions regarding a flow are either to accept (want) or reject (don't want) the flow. However, classifying traffic into wanted or unwanted may not always be possible or even desirable. For instance, the decider may not have sufficient information to accept or reject the first packet from a new client, or the decider may choose not to classify traffic when it is not under attack. In these cases, the decider may leave the traffic as unclassified.

By definition, all capability architecture implementations must have these three classes, but some may have additional sub-types. For example, PATRICIA [42] defines two sub-types called *control* and *authorized* within the wanted traffic class. We do not consider sub-types in the taxonomy since they are mostly implementation specific. However, we note that implementations may chose additional traffic classes in order to discern traffic to different destinations. On the other hand, more traffic classes means the decider

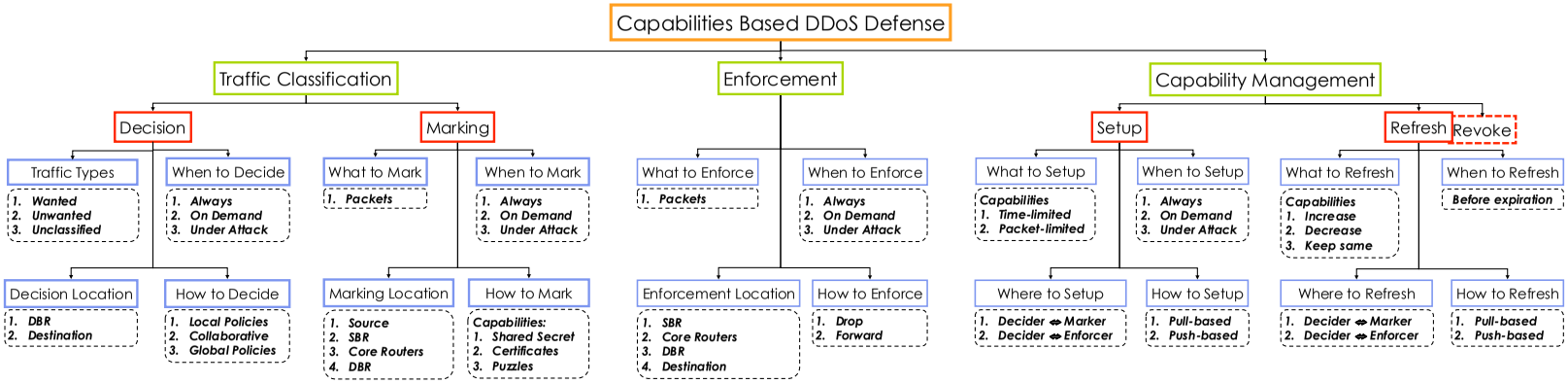


**Figure 3.1.2:** The Taxonomy of Capability Architectures

may need to expend more resources when classifying traffic. The implementation must weight the tradeoffs associated with the increased overhead versus the benefits gained from adding more traffic classes.

#### 3.2.1.2 Decision Location

Location of the decider determines sensitivity/correctness of classification. In most cases, decisions made at the destination are more accurate, but there are other locations and factors to consider. We present a few arguments for choosing the location for the decider.

- *Correctness* – Two factors influence correctness: i) knowledge of resources under attack and, ii) knowledge of receiver's policies. This knowledge is more available closer to the destination, i.e., traffic wanted by the destination, traffic in accordance with the destination's network policies, and congestion information. Specifically, the destination has most knowledge about its own policies and its resource limits, while the destination border router ($DBR$) knows more about the network policies and condition of the narrow access link. These two locations make good choices for a decider, but beyond them, correctness is difficult to achieve. The core network may have knowledge about congestion, but does not know the receiver's policies.

- *Deployment Issues* – Traffic classification requires installing a classifier module at the decider. At the destination host, this may be a simple software upgrade, but upgrading a router such as the $DBR$ is more complex. However, deploying the classifier on the $DBR$ benefits multiple destinations downstream from the $DBR$, whereas installing it on the destination benefits only that one host. The same deployment issues apply when considering marking and enforcement locations.

- *Effect on Resources* – The classification process consumes cpu and memory resources proportional to the number of flows reaching or transiting the decider. Since a $DBR$ transits traffic for multiple destinations, it needs to commit more resources compared to a host that only needs to keep track of its own flows. These additional constraints play a role in choosing the decider as well.

#### 3.2.1.3 How to Decide

Deciding what traffic to accept or reject requires inputs/information at the decider. These inputs could originate locally, entirely based on *local policies*, they may originate from a distributed entity based on *collaborative policies*, or may originate from a third-party system using *global policies*.

More specifically, in local policy based systems the decider may maintain a local reputation value for clients, or a blacklist of misbehaving clients. Other mechanisms include, a local intrusion detection system (IDS), CAPTCHAÕs to distinguish human users from automated bots, client authentication, and techniques based on computational puzzles [34] and so forth. Alternately, the decider may coordinate with a group of external or internal nodes (including other deciders) which provide additional information in making decisions. We call these collaborative policies. For example, the decider could make decisions depending on input from a collaborative reputation system, or a collaborated blacklist of misbehaving clients. However, in a distributed network different nodes have different interests, so collaborative systems need to address issues with agreement, trust and security between the participants. Finally, the decider could use a service from a third party system to make decisions. Examples include third party reputation systems, global blacklists of misbehaving clients and/or lists with unallocated address blocks.

#### 3.2.1.4 When to Decide

An interesting question we raise in the taxonomy is when to invoke the decision process. Although decisions can be made at all times, the process itself consumes resources (cpu, memory), and conserving these resources might be important for some systems. In the taxonomy, we define three possible choices i) decide *at all times*, ii) decide *when under attack*, and iii) decide *on demand*. The later two choices reduce the overhead when there are no ongoing attacks. However, they might raise some new challenges; for instance, how does a decider detect an ongoing attack? How quickly can the decider respond to attacks? Furthermore, initiating the decision process on demand requires authentication between the decider and the requesting party, which further complicates the overall architecture.

These three choices also apply to marking (i.e., when to mark), enforcement and setup, but we omit the discuss in later sections since they raise similar issues.

### 3.2.2 Traffic Classification: Marking

The second part of the traffic classification process is to mark packets with capabilities. Marking allows the decider to convey its decisions to the network.

#### 3.2.2.1 What to Mark

Marking adds capability bits to *packets*. Although, conceptually it is possible to mark flows, we do not consider this case in the taxonomy. Marking a flow would mean that the same capability must be used for an unspecified number of packets, which contradicts requirements such as periodic refresh and making a change in decision.

#### 3.2.2.2 Marking Location

Although decisions are made closer to the destination, marking closer to the source is more effective since it allows the network to take enforcement actions early. However,

in the taxonomy we consider several locations for marking and offer some insights in choosing these locations.

- *Effectiveness* – Marking at the source gives the most opportunity for the network to take enforcement actions, i.e., it enables every location from the source up to the destination to enforce. The $SBR$, core and $DBR$ may also mark packets to achieve compliance with their own network policies. For example, an $SBR$ may mark on behalf of its hosts to filter certain clients from communicating with the destination. The effectiveness of enforcement reduces as marking is moved closer to the destination.

- *Security* – Although the source is the most effective location to mark, it is also the least trustworthy location. The host software may be insecure and may be compromised. The attacker could use the source to flood the network after inserting random bits and attempting to pass them as valid capabilities. This will increase the processing overhead at the enforcers. On the other hand, the $SBR$, core and the $DBR$ may not suffer from the same security issues as the source, since router software is much different, and moreover these are well managed compared to a source.

- *Incentives* – The source has the most incentive to mark since it wants its traffic to reach the destination. Similarly the $SBR$ may also mark to achieve policy compliance or to save uplink bandwidth in case its network contains compromised hosts (example: DSL, cable operators). Similar arguments apply to core networks, but the operators in these networks may be hesitant to add tasks to the routers beyond basic forwarding. We believe incentives need to be strongly considered when choosing a marking location.

#### 3.2.2.3 How to Mark

How to mark refers to the techniques used to construct, manipulate and test capabilities. Fundamentally, a capability is a secret between the decider, marker and enforcers, and the adversary should not be able to reconstruct/guess it easily.

A straightforward approach for constructing, manipulating and testing the capability is to use *cryptographic* techniques. For example, TVA [48] uses keyed hash functions, PATRICIA [42] uses digital signatures in request packets and PortCullis uses cryptographic puzzles. We refer to them collectively as cryptographic capabilities. Alternately, if any or all of the operations to construct, manipulate, and test capabilities can be achieved without using cryptography, we refer to them as *non-cryptographic* techniques. Cryptographic techniques may different requirements and properties; for instance digital signatures require public key infrastructure (PKI), where as keyed hash functions in TVA do not require such an infrastructure. On the other hand, capabilities created from keyed hash functions in TVA are path dependent, while digital signatures are path independent. Non-cryptographic techniques are hard to envision, since most existing capability approaches utilize cryptographic techniques. We discuss this in more detail in Section 3.4.

### 3.2.3 Enforcement

Enforcement takes forwarding actions depending on the capability in a packet. We refer to the node taking the enforcing action as the *enforcer*. Note that enforcement actions are taken on individual packets (and not flows), since marking attaches capabilities to individual packets.

#### 3.2.3.1 Enforcement Location

The enforcement location determines how quickly unwanted traffic is removed from the network. We consider the SBR, core routers, DBR and destination for enforcement

location.

- *Effectiveness* – Removing unwanted traffic closer to the source is more effective since it saves network resources and prevents the attacker from consuming bottleneck resources in the network. In this regard, the SBR and core routers are likely candidates, but the DBR and the destination could also enforce to ensure unwanted traffic is entirely removed. However, removing traffic at the DBR and the destination is not as effective as removing at the source or in the core.

- *Incentives* – The source has no incentives to enforce; especially if it is compromised. The $SBR$ on the other hand has incentives to block unwanted traffic from leaving its network, considering the source network might be paying for the network service. Similarly, the $DBR$ has more incentive to block unwanted traffic from entering its own network to protect its hosts. Core routers have fewer incentives; they may save some network resources, but the complexity of doing so might out weigh the benefits of removing traffic. However, they are an important location nonetheless. Providing incentives for the core network is an important problem.

#### 3.2.3.2 How to Enforce

The forwarding action taken during enforcement could be to: i) *drop* a packet, ii)*forward* it, or iii) *rate limit*. Conceptually, enforcement starts with verifying the capabilities inside the packet. If the verification succeeds the packet is delivered to a scheduler, which takes further action such as forwarding it to the appropriate outgoing interface. If verification fails, the packet is either dropped immediately or delivered to the scheduler, which may then take further actions such as rate limit the packet (to reach a certain target rate), or delay the packet for a short duration to send it when resources become available, or simply drop it. Note that, although we define only three basic actions

here, it is possible for an architecture implementation to define additional scheduler actions such as increasing the priority of wanted traffic, demoting a packet that failed verification and so forth.

### 3.2.4 Capability Management: Setup

Setup initializes capabilities between the decider, marker and the enforcers. Figure 3.1.2 shows capability management with setup and refresh. Note that the definition of setup in this context is different from the general definition of setting up a new connection (for example, a TCP connection). The later may occur multiple times between a source and a destination without requiring additional capability setups.

#### 3.2.4.1 What to Setup

In the taxonomy we identify two types of capabilities: i) *time-limited*, which are valid for a specific amount of time, and ii) *packet-limited*, which are valid for specific number of packets. The length/period of validity of a capability is an important tuning parameter – a capability that is valid longer may authorize several packets, which is harmful if an attacker gets hold of the capability. While a capability with shorter period can only authorize fewer packets, but needs to be refreshed more often, which increases overhead. Note that capabilities might be both time-limited and packet-limited. For example, in TVA, capabilities are both time-limited and packet-limited, while in SIFF, capabilities are only time-limited. Packet limited capabilities require additional state at the enforces to keep track of the authorized packets.

#### 3.2.4.2 How to Setup

Determining how to setup involves understanding how the decider distributes capabilities to the marker and enforcers. In the taxonomy, we consider two very different communication models for setting up capabilities: i) *Pull* capabilities from the decider,

or ii) *Push* capabilities to the marker.

In the pull approach the marker requests capabilities from the decider. To do that, it must first *discover* the decider (since decider could be different from the destination) and then *retrieve* the capabilities. To learn about the decider the marker may for example, query an external database such as DNS, or send a control message towards the destination asking for the decider. Note that if the decider is the destination itself, then the discovery process is simple since the marker already knows the destination address. Following discovery, the markers and enforcers use architecture specific mechanisms to retrieve capabilities. For example, in TVA the source sends an explicit capability request packet, while the enforcers implicitly learn about the capability as part of the setup (they generate bits of the capability token). However, prior work has shown that this type of retrieval process is susceptible to denial of capability (DoC) attacks [7, 34]. Additional mechanisms are needed to prevent DoC attacks in the pull model [34].

In the push approach the decider pushes capabilities to the marker. However, in order to do so, the decider must first discover the marker and enforcers associated with a source and then transfer the capabilities to them. Similar to the pull approach, the decider may use DNS, or control messages to learn about the marker (and enforcers), and use architecture specific mechanisms to transfer capabilities. Unlike the pull approach, the push approach does not suffer from DoC attacks since the source no longer needs to ask for capabilities using unprotected request packets. To our knowledge, none of the existing capability mechanisms use the push model to setup capabilities. We discuss more details of the pull mechanism in Section **??**.

### 3.2.5 Capability Management: Refresh

Authorizations issued by the decider cannot be permanent, otherwise a source with valid authorizations may be compromised and may become a threat. To ensure continued communication between the ends, the decider needs to refresh capabilities periodically.

#### 3.2.5.1 What to Refresh

The capabilities established during setup need to be refreshed for two reasons. First, decisions are not static but may change over time. For instance, a change in decision might be triggered when a source becomes malicious. Second, the marker or the enforcers may change as a result of path changes or node failures. To accommodate these changes the capabilities need to be refreshed. The decider may choose to *increase*, *decrease*, or *keep the same* number of capabilities at the time of refresh. A special case of decrease is to issue no capabilities at all, which essentially terminates the authorization given to a source. The capabilities issued during refresh may be time-limited or packet-limited.

#### 3.2.5.2 How to Refresh

The decider could use different mechanisms to refresh capabilities; it could issue new capabilities in each refresh cycle, or change the state associated with a capability at routers. Specifically, if we assume each capability initially permits $N$ packets, the decider can request the routers to permit $N + n$ packets to cause an increase, or request $N - n$ to decrease the number of packets permitted per capability. The capabilities can be refreshed through a push or a pull based communication process similar to setup.

#### 3.2.5.3 When to Refresh

Since capabilities expire after a while, they can be refreshed *anytime before they expire*. However this raises two important issues: 1) the overhead involved in periodic refresh, and 2) the maximum time before a change can take effect. These two issues are effected by the lifetime of a capability and the duration before which a capability is refreshed. Longer expiration periods cause slower response to decision or node changes (decider must wait until existing capabilities expire), while shorter expiration periods increase overhead. Alternately, if capabilities are refreshed much before they expire, changes can take effect sooner, but it also increases overhead. In addition to these factors, the

decider needs to consider round trip time and message loss to determine the maximum duration it can wait before refreshing capabilities.

### 3.2.6 Capability Management: Revocation

An alternative to the (soft-state) refresh mechanism is an explicit (hard-state) *revocation*. Although refresh and revocation are solutions to the same problem, to our knowledge existing capability implementations largely use the refresh mechanism. We now describe the tradeoffs involved in choosing either the refresh or the revocation mechanism to remove capabilities from the network. Specifically, the tradeoffs we discuss include, delay in response to changes, and overhead.

Unlike refresh, revocation responds quickly to changes since the dominant delay factor is the propagation time for the revocation message to reach the marker and enforcers. However, to revoke capabilities the decider needs to know the markers and enforcers, and moreover revocation requires additional messages from the decider. These two factors contribute to the overhead in revocation. On the other hand, the delay in response to changes in refresh depends on the lifetime of a capability. Shorter lifetimes decrease this delay, but increase management and message overhead. For example, with shorter lifetimes, the destination needs to update the marker frequently. But the advantage of soft-state refresh is that it does not require the decider to know about all the enforcers in the network.

## 3.3 Evaluation of Proposed Implementations

Using the taxonomy framework, we compare several capability implementations and point out key differences. Our evaluation covers SIFF [46], TVA [48], PATRICIA [42], Fine Grained Capabilities (FGC) [26] and PORTCULLIS [34]. Figure 3.3.1 shows the evaluation results, which points out several unexplored areas (marked in bold).

| | | | | SIFF | TVA | PATRICIA | FGC | PORTCULLIS | **Unexplored** |
|---|---|---|---|---|---|---|---|---|---|
| Traffic classification | Decision | Traffic types | Wntd | Privileged | Regular, Request | Control, Authorized | Packets with reputation ≥ $R_{MAX}$ | Requests with puzzle level > *k* | |
| | | | Unwntd | Invalid capability packets | Demoted | Demoted | Packets with reputation ≤ $R_{MIN}$ | Requests with puzzle level < *k* | |
| | | | Unclsfd | Unprivileged | Legacy | Regular | Packets with reputation ≅ $R_{MID}$ | | |
| | | Where | | Destination | Destination | Destination, DBR, SBR | Destination | All except source | |
| | | How | | Local policies (unspecified) | Local policies (blacklists, Captchas) | Local policies (blacklists) | Local policies (reputations, blacklists) | Local policies (puzzles) | **Collaborative, Global Policies** |
| | | When | | Always | Always | Under attack | Always | Under attack | **On Demand** |
| | Marking | What | | Packets | Packets | Packets | Packets | Request packets | |
| | | Where | | Source | Source | Source | Source | Source | **SBR, Core, DBR** |
| | | How | | Cryptographic(hash) | Cryptographic(hash) | Digital Signature, Cryptographic(hash) | Cryptographic(symmetric key) | Cryptographic(puzzles) | **Non-Cryptographic** |
| | | When | | Always | Always | Under attack | Always | Under attack | **On Demand** |
| Enforcement | | What | | Packets | Packets | Packets | Packets | Packets | |
| | | Where | | All except source | All except source | All except source | DBR, Destination | All except source | |
| | | How | | Immediately drop unwanted | May drop unwanted | May drop unwanted | May drop unwanted | Immediately drop unwanted | |
| | | When | | Always | Always | Under attack | On demand | Under attack | |
| Management | Setup | What | | Time limited | Time limited, packet limited | Time limited | Time limited | Packet limited | |
| | | How | | Pull | Pull | Pull (layered) | Pull | Pull (layered) | **Push** |
| | | When | | Always | Always | Under attack | Always | Under attack | **On Demand** |
| | Refresh | What | | Increase, decrease, keep same | Increase, decrease, keep same | Increase, decrease, keep same | Increase, decrease, keep same | | **Revocation** |
| | | How | | Issue new | Issue new | Issue new | | | |
| | | When | | Before expiration | Before expiration | Before expiration | Before expiration | | |

**Figure 3.3.1:** Evaluation of various capability architecture implementations

### 3.3.1 Traffic Classification: Decision

As noted in the taxonomy, the decision process involves determining whether a flow is wanted or unwanted. We now discuss how the various capability implementations achieve this task.

#### 3.3.1.1 Traffic Types

The implementations define all three traffic types described in the taxonomy (albeit, with different terminology). However, some crucial differences exist. SIFF treats all packets without capabilities as unclassified traffic, including capability request packets which do not have any capabilities. While all the other implementations classify request packets as some form of wanted traffic. TVA uses fair queueing, PATRICIA uses digital signatures, PORTCULLIS uses puzzles, and FGC uses reputation values to identify wanted request packets. For example, PORTCULLIS treats all request packets with a puzzle solution of difficulty greater than $k$ as wanted traffic.

Within the wanted class, TVA, FGC and PATRICIA apply queueing algorithms to prioritize different flows. However, as we noted in the taxonomy, the enforcers (routers) need to expend more resources with increasing packet classes. The impact of which has not been evaluated in the implementations.

#### 3.3.1.2 Decision Location

The destination is the decider in all implementations, but interestingly PATRICIA involves both the DBR and the SBR in the decision process. However, these nodes are only responsible for making decisions on request packets. The SBR decides if a source is allowed to communicate with the destination (based on its network policies), while the DBR decides if the destination is allowed to accept requests from certain networks. In PORTCULLIS, decisions regarding request packets are made at all location except the source, since the request capacity is a limited resource at every link.

#### 3.3.1.3 How to Decide

The implementations use local policies to make decisions. Collaborative and global policy systems have not been considered in the current implementations. An example of a collaborative system can be found in [24].

#### 3.3.1.4 When to Decide

In SIFF, TVA, and FGC, the destination always decides regardless of whether it is under attack or not. In PATRICIA, the destination only decides when it is under DDoS attack. Similarly, in PORTCULLIS, routers require (and check) puzzles in request packets when the destination is under attack. Deciding on demand is not considered in any of the implementations we evaluated.

### 3.3.2 Traffic Classification: Marking

There a few key differences in how the implementations mark packets, which we discuss next.

#### 3.3.2.1 Marking Location

The implementations mark at the source, mainly due to its effectiveness. However, as noted in the taxonomy there are other possible locations for marking which have not been explored.

#### 3.3.2.2 How to Mark

All implementations we evaluated use cryptographic techniques to construct, manipulate and test capabilities. Specifically, SIFF, TVA and PATRICIA construct and test capabilities piece-by-piece at each router using keyed hash functions. In FGC, the capability is a symmetric key shared between the source and the destination. In PATRICIA, the SBR

marks request packets with a digital certificate, and in PORTCULLIS, the source solves strong cryptographic puzzles and includes the solutions in request packets.

### 3.3.3 Enforcement

The following is the evaluation of enforcement.

#### 3.3.3.1 Enforcement Location

Most implementations enforce at all locations except the source. FGC, on the other hand, enforces at the destination and DBR. However, FGC requests core routers to enforce when enforcement at the DBR and destination is alone not sufficient to thwart attacks.

#### 3.3.3.2 How to Enforce

Wanted traffic receives higher priority than other traffic, so it is always forwarded when sufficient bandwidth is available. Unwanted traffic is always dropped in SIFF, but not dropped immediately in TVA, PATRICIA and FGC (it may be dropped upon reaching a congested link). Unwanted request packets are always dropped in PORTCULLIS. A benefit of forwarding packets that fail to verify is that when network paths change, the capability may become invalid at certain enforcers. If the enforcer simply drops the packet the destination may take a while detect and recover from the error, but if instead the enforcer forwards it, the destination may detect the problem and trigger an immediate refresh cycle.

#### 3.3.3.3 When to Enforce

SIFF and TVA enforce at all times, PATRICIA enforces only when the destination is under attack, and FGC enforces on demand. A careful evaluation is needed to understand the impact of these choices. We believe routers may reduce some overhead using the on

demand or under attack approach, but they may also allow some unwanted traffic to get through.

### 3.3.4 Capability Management: Setup

We find that capability management is not well defined in most implementations. However, there are some details that set the implementations apart.

#### 3.3.4.1 What to Setup

SIFF, FGC and PATRICIA implement time-limited capabilities. TVA implements packet-limited capabilities. PORTCULLIS puzzles are equivalent to packet-limited capabilities (one puzzle solution permits one request packet). Verifying packet limited capabilities requires additional state at routers, which is an important resource to protect.

#### 3.3.4.2 How to Setup

To setup capabilities, SIFF, TVA, PATRICIA, FGC and PORTCULLIS use the pull model, but PATRICIA implements a two-layer pull mechanism in which the marker requests its SBR for an initial capability (to include in request packet), and uses it to request subsequent capabilities from the destination. PORTCULLIS also implements a two-layer pull mechanism, but with puzzles. The puzzle solutions in the request from a source is treated as a capability, which in turn is used to request subsequent capabilities from the destination.

### 3.3.5 Capability Management: Refresh

The implementations do not provide sufficient detail in this area, but we believe they support increasing, decreasing or keeping the same number of capabilities in a refresh cycle. To also push new capabilities to the source to cause an increase or to keep the same number of capabilities.

## 3.4 Findings from the Taxonomy

Although we did not uncover new open problems other than what has already been pointed out, such as the DoC attacks [7,34], and the colluding attacker problem [42,48]. We believe there are a few missed opportunities that are worth exploring. We now discuss these in more detail.

### 3.4.1 Alternate Decision and Marking Locations

Current capability architectures involve end hosts in setting up capabilities. However, a malicious source may insert bogus capabilities, share its capabilities with others, or steal capabilities issued to a different source. In other words, the source is the least trustworthy location to mark packets. Similarly, a compromised host may authorize a large volume of traffic to its network and cause harm to other hosts in the same network (colluding attacker).

**Marking Locations**: Marking packets at locations other than the source may address some of the issues with malicious hosts. For instance, if marking is done at the source border routers, a compromised host can no longer easily steal or insert bogus capabilities. The core routers and destination border router may also mark, but the effectiveness is reduced as marking becomes distant from the source, and hence the opportunity to remove unwanted traffic much earlier in the network. Marking at the SBR requires that the decider know about the SBR, or have some mechanism to discover the SBR (i.e., if multiple SBR exists, the decider needs to know where to send the capabilities).

**Decider Locations**: Although the decider location determines correctness in the decisions made regarding a source, and decisions are more accurate at the destination, there are several alternatives to consider including destination border router, core routers and the source border routers. Network capabilities fundamentally distinguish between

wanted and unwanted traffic. An attacker, however, may be able to use colluders at appropriate places to drive shared bottlenecks into saturation and squeeze out the target destination's traffic without directly attacking the destination. To address this issue TVA proposes fair queueing all wanted traffic, but this is a fragile solution, as the attacker can employ many colluders, effectively neutralizing the FQ controls. PATRICIA involves the DBR in decision making, but this approach works only if the attacker is located within the DBR's network. An alternative is to use collaborative or global decision systems such as multiple DBRs participating in the decision process. However such systems are very hard to implement and deploy.

### 3.4.2 Non-Cryptographic Capabilities

Cryptographic capabilities are hard to guess or forge. However, they consume cpu resources at both end-hosts and routers, and these resources may be critical at a few network points (such as congested routers). Moreover, they may require changing the router architecture to support the new cryptographic operations. Using non-cryptographic techniques would alleviate these issues, but they must be hard to guess or forge. We sketch a couple of examples next. One possibility is for the destination to use a state based approach in which the destination creates the capability and distributes it to key enforcers and the source. The enforcers simply cache the capability and verify it against the one they receive from the source (using simple matching). Another possibility is for the capability itself to dictate the forwarding path to the destination. The decider provides an encoded source route as the capability, and without it packets from a source cannot reach the receiver.

### 3.4.3 Push Capabilities To Marker

A pull model for setting up capabilities is susceptible to DoC attacks. PORTCULLIS provides a solution to defend against these attacks. However, an alternative is to use a

push model for capability setup. To sketch an example, suppose we disallow the sender from requesting capabilities directly from the decider. Instead, the decider periodically pushes a few initialization-capabilities to a number of trusted intermediaries, say, the SBRs. The senders are required to obtain these initial capabilities from the SBR, so they may send requests to the receiver and obtain additional capabilities for subsequent traffic. The SBRs work in conjunction with the decider ensuring that offenders are not granted initial capabilities. For instance, the SBRs could install a local blacklist. To attack this system, the attacker may flood the SBR, but the SBR can easily detect such attacks and isolate the compromised hosts in its own network. An intelligent attacker may use several attack sources to consume all available initialization capabilities from an SBR, but since the decider periodically pushes capabilities, the initialization capabilities are restored shortly. Moreover, the SBR may now categorize these sources as offenders, and either refuse or reduce the number of initialization capabilities granted to these sources.

### 3.4.4 Locating the Enforcers

To verify capabilities the decider needs to signal enforcers. In single path networks the enforcers are located on the path between the source and destination. However, all current designs tie the capability to the path, which will not work in networks where the path changes frequently (e.g., mobile networks). Thus, according to our taxonomy, the enforcement function may have to be easily movable to address many networks. Such ability will also be useful if there are frequent routing changes.

### 3.4.5 Revocation

Explicitly revoking the capabilities allows the destination to quickly cutoff clients that turned into attackers. However, the current implementations do not explore revocation. To see the importance of revocation, imagine the destination sending a revocation mes-

sage for a particular capability to the enforcers, which then temporarily maintain a state to block packets at a very fine granularity, such as packets using that specific capability. When the capability itself expires, the state can be removed. Our taxonomy identified revocation as an important new direction to explore in capability architectures.

## 3.5 Discussion

We studied capability architectures in great detail not only to understand the problem and solution space but also to motivate our own design. We found that capability architecture design space is generally rich and mostly complete. The two important problems in capability architectures, i.e., the colluding attacker problem and the denial of capabilities problem have been studied and solutions were proposed. However, these solutions are not without additional constraints. For instance, fair queueing all wanted traffic to address the colluding attacker problem (as suggested in TVA) requires additional processing and state at routers. The state required to classify packets is proportional to the number of flows and the router needs to process every packet to classify its flow. In addition, the solution proposed in TVA cannot easily address the case when multiple colluders game the system into granting more share of the resources to the attacker. The alternative we suggested in the taxonomy to involve different decision locations (core routers, multiple DBRs) must deal with cooperation among the parties involved. This may be difficult to achieve given distributed entities may not always agree on a common outcome. We believe that colluding attacks are possible because the attacker knows where the destination is located. A location hiding based approach that completely hides the destination address may not have this problem.

Another important issue with capability architectures is that deploying capabilities requires making fundamental changes to the current Internet architecture. The packet header must now carry a variable length capability that the routers have to verify.

Routers are required to perform cryptographic operations to verify the capabilities. Processing the packets at the same line rate as current Internet routers means the routers in capability architectures need faster CPUs or specialized processors, or in other words, routers must be upgraded and their software and hardware architecture may need to change. Moreover, host software and middleboxes such as proxies and home routers must be upgraded to support the new model. Due to the number of devices that only work with IP, deployment may become difficult.

Although our study of capability architectures has not found new open problems, we found some new directions that are worth exploring. We applied these lessons in our novel location hiding architecture, which we present in the next chapter. Specifically, we prevent the colluding attacker problem by hiding the destination address. We achieve this location hiding through non-cryptographic based *hidden paths* that are similar to capabilities, i.e., if a host does not have a hidden path, it cannot send any traffic to the destination. Our hidden paths require some software changes to the routers, but do not change the IP packet structure or the router architecture. In addition to this we decouple the entities marking packets from the source. In our architecture, intermediate proxy nodes mark packets with hidden path on behalf of a source (i.e., an alternative to marking at the source). Moreover, in our architecture the destination explicitly sets up (i.e., pushes) hidden paths to the proxies without ever exposing its IP address to the proxies. Finally, our architecture has both a soft state approach to remove hidden paths as well as an explicit hidden path removal protocol to cutoff a proxy from sending any traffic to the destination. In other words, the reaction time to cutoff a misbehaving proxy is in the order of a round trip time.

# Chapter 4

# The Epiphany Location Hiding Architecture

In this chapter we present the Epiphany location hiding architecture. We first discuss the challenges facing location hiding and describe Epiphany using an example communication process. We then present our attack model and discuss the various components of Epiphany in more detail, namely the proxies, destination and hidden paths.

## 4.1 Overview

We start with an overview describing the challenges facing location hiding, how Epiphany addresses those challenges and give an example of the communication process.

### 4.1.1 Challenges in Location Hiding

In order to effectively defend against DDoS attackers, location hiding architectures must address a few important challenges. We outline these here so the reader can establish the basis for why we made certain design choices in Epiphany.

- *Location hiding architectures need to ensure that the destination remains hidden despite compromised or malicious proxies*. Minimizing the number of trusted components, carefully controlling what information is released to the proxies, and controlling how the proxies reach the destination are important factors to consider when hiding the service location. Otherwise, once the destination's address is out, the attacker will circumvent the proxy layer and directly target the destination. To avoid such potential threats, SOS [17] and Mayday [5] suggest putting a defensive

filtering perimeter around the destination, so that even if the destination address is revealed to the attacker, the damaging traffic can be limited at this perimeter. However filtering presents yet another set of challenges; for instance, the filtering perimeter must be large enough to cover bottlenecks near the destination network.

- *Location hiding architectures need to make the service resilient in the face of attacks on the proxies*. Since the destination is no longer a direct target, the attacker will target the proxies instead. One approach to dealing with attacks on the proxies is to use a large number of proxies, and change them periodically to evade the attacker. However, in order to quickly recruit and deploy large numbers, the proxies must be lightweight, i.e., they need not require special hardware, software or store sensitive user information. Recruiting proxies from end-hosts such as peer-to-peer networks is one way to get them in large numbers, but they are generally not trustworthy.

- *Location hiding architectures need to assign and manage proxies in such a way that collateral damage to legitimate clients is limited*. Although increasing the number of proxies may present a very wide target for the attacker, a legitimate client using a proxy to reach the destination may still be affected if that proxy is attacked. In fact, the attacker may selectively target specific proxies to disable or disrupt portions of the proxy network. The destination needs to assign and manage proxies in such a way that collateral damage to legitimate clients is minimized in the presence of large scale DDoS attackers.

- *Lastly, location hiding architectures need to have a scalable and robust discovery mechanism for the clients to learn about the proxies*. To reach the service, clients need to learn about the proxies using the discovery mechanism. However, this discovery service must be able to cope with frequent changes in the proxy set, and

moreover, it must be robust against attacks; otherwise the discovery service will become a single point of failure for the entire system.

### 4.1.2 Salient Features of Epiphany

In Epiphany, we address these challenges with some simple but novel set of ideas such an anycast and multicast routing. Epiphany has the following distinguishing features.

- *Strong Location Hiding*: Epiphany completely hides the destination from everyone in the network, including the sources and the proxies. Instead, the proxies are given revokable *hidden paths* to reach the destination. Therefore, there is no need to trust any proxy or employ defensive filtering perimeters. Moreover, if a proxy becomes compromised the destination location is not revealed to the attacker. If the destination is able to identify the misbehaving proxy, it can cutoff the hidden path to the proxy to prevent it from causing any collateral damage to traffic from other proxies.

- *Localize Attacks*: Epiphany separates setup proxies (SPs) from data proxies (DPs), and employs IP anycast on the SPs to create distinct network regions. Requests generated in a region are automatically directed to the nearest local SP by the underlying routing system. If a network region created due to an anycast SP does not have any attacker bots, clients from that region will be unaffected by bots in other regions, as these bots have no way of sending requests to that local SP. On the other hand, if a region has both attackers and legitimate clients, more SPs could be instantiated in that region to further break it down into smaller regions. These smaller regions may in turn have no attackers, or have fewer attackers that are easier to locate and disable. In other words, Epiphany localizes attackers using anycast SPs.

- *Trust Based Segregation*: Although the SPs are public, the DPs are not made public to everyone in the network. A client can send a request through an SP and ask the destination to provide access through a DP. If the destination does not trust the client, it will not reveal a DP to the client. On the other hand, a client that is trusted by the destination is given access to a DP that no other client may know about. The communication of this important client will not be affected by ongoing attacks against the system (unless the attacker is able to determine the address of the DP that the legitimate client is using).

- *Fine-Grain Traffic Control*: An authorized source in Epiphany does not have unrestricted access to the destination. The source must comply to the destination's requirements for sending traffic, i.e., it must comply to a certain sending rate, otherwise proxies perform rate limiting to prevent excess traffic from reaching the destination.

- *Robust Proxy Discovery*: The client only needs to learn a small number of proxy addresses due to the use of IP anycast. Moreover, these addresses have a long life-time even thought the underlying proxies may change frequently. Epiphany leverages existing DNS to make proxy discovery robust. Proxy discovery for DPs involves the destination directly telling the source which DP to use, so there is no burden of making them available on DNS.

### 4.1.3 A Sample Communication Process

To better understand the interaction between the components, we present an example of the communication process between a source S and the destination D when there is no ongoing attack. Figure 4.1.1 shows the Epiphany architecture. Here D's address is hidden from S and the only way to reach it is through a number of proxies. Some proxies, designated as *setup proxies* (SP) are responsible for forwarding requests from

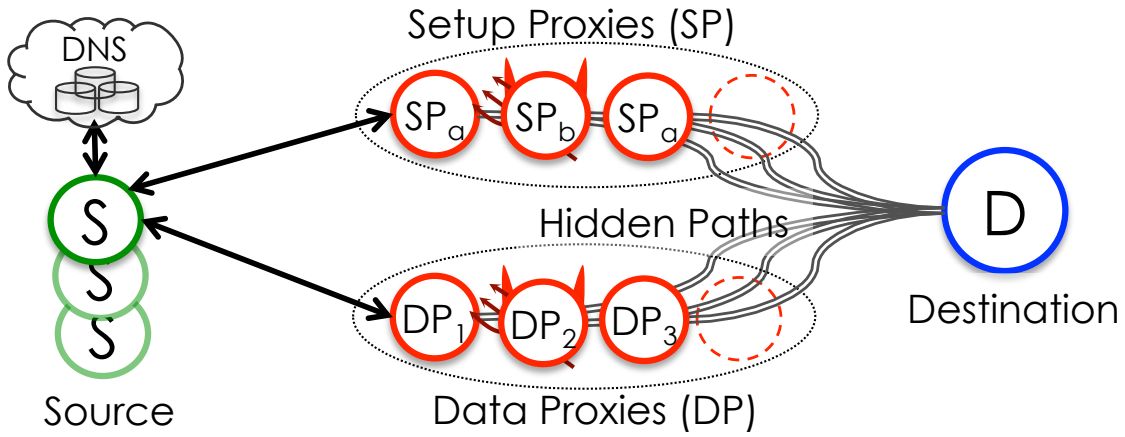


**Figure 4.1.1:** Epiphany architecture and components

the source. The source learns about these from DNS. The destination assigns a different set of proxies called *data proxies* (DP) in order to communicate with the source. For ease of exposition, we break down the communication process into three phases as shown in Figures 4.1.2, 4.1.3 and 4.1.4.

#### 4.1.3.1 Pre-Setup

Before any source can communicate with the destination, D must first select some SPs. We assume the destination learns of the possible SPs through some external means; for instance, SPs may run on user machines, be available as part of a paid service, or are setup by network operators as a service to their users. We assume that SPs have a public anycast address in addition to their unicast address, but the latter is not made public. Different SPs may have different anycast addresses, so the SPs will form distinct anycast groups. Once the destination selects its SPs, it is provided with the unicast addresses of the SPs. We assume only the destination knows about the unicast addresses of the SPs and that these unicast addresses are safely communicated to D when it learns about the SPs.

Note that the above assumes an instance of Epiphany that is not available to the general user to deploy a service (although any user can access the service). While this is a

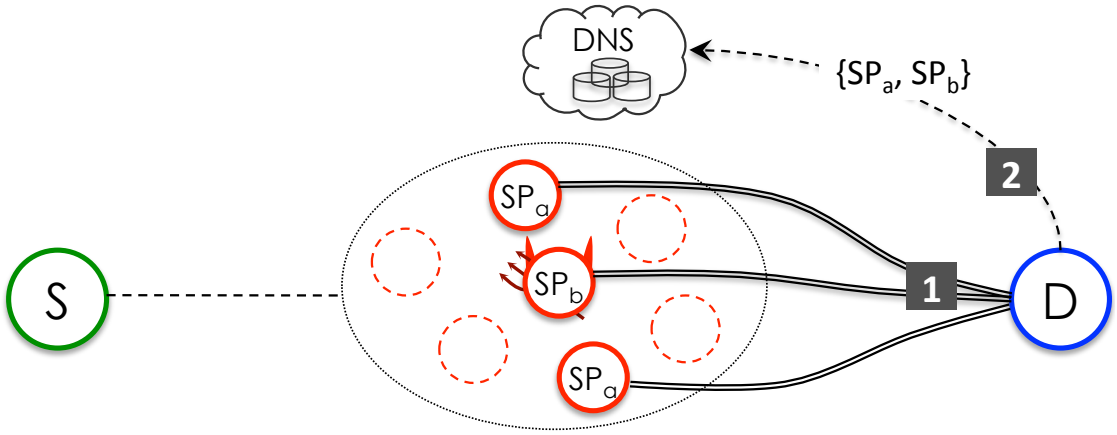


**Figure 4.1.2:** Pre-Setup phase

reasonable assumption for many critical services (they are deployed by authorized users only), there is nothing to stop Epiphany from being used by general users to provide a service. In this case, however, without some access controls, it becomes relatively easy to infiltrate the SP network and enumerate the unicast address of the SPs and attack the service, a weakness shared by other similar services such as SOS and TOR. While solutions to this problem are beyond the scope of this work, we note that any solution that works with other peer-to-peer services will also work for Epiphany.

The destination D uses the unicast addresses to build hidden paths to the SPs (Section 4.3) and provides an identifier for the corresponding hidden path to each of the SPs. The SPs can use this hidden path identifier to forward requests to D. The SPs are also assigned a service-controlled request rate limit. The rate limit may depend on the aggregate users the service can sustain, or be tailored to the path capacity from an SP to the destination. Finally, D publishes the anycast SP addresses in DNS (Section 4.2.1). Note that further adjustments to the SP population can be done dynamically and transparently without disruptions to the service. However, we assume D has a secure channel to update the DNS if changes to the SP records are necessary; for instance, D could construct hidden paths to certain intermediary hosts that allow D to securely access the DNS.

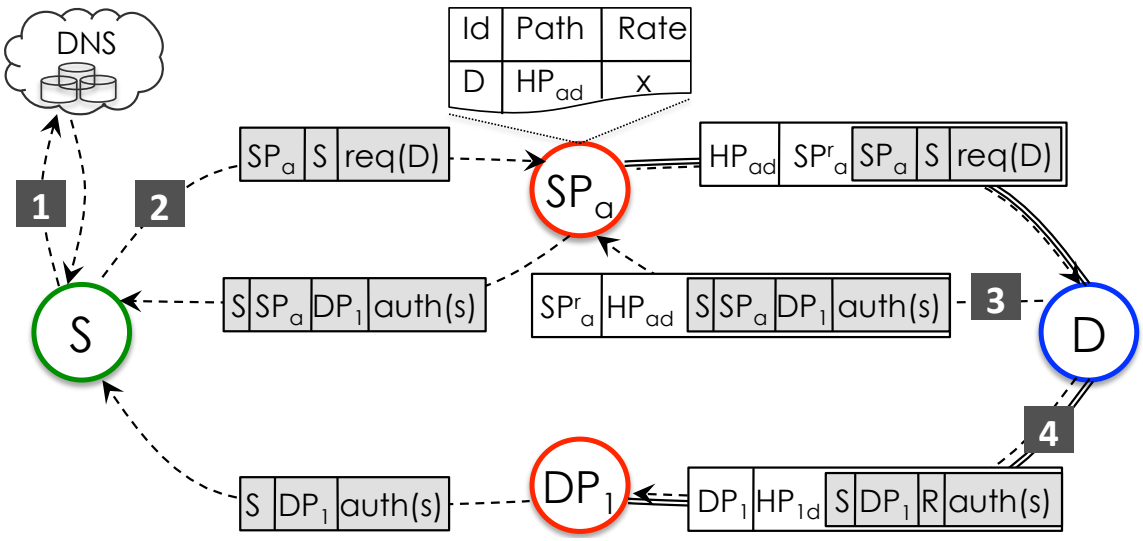


**Figure 4.1.3:** Setup phase

#### 4.1.3.2 Setup

Now that the destination has provided some SPs, when the source wants to access the service, it consults DNS to get the anycast addresses associated with the service. The source chooses an SP (if multiple anycast addresses are involved) and sends an Epiphany request for D. The request may include identification or authentication information about S that the destination can verify (for instance, a prior authorization token, a reputation value, a digital certificate). Figure 4.1.3 shows this process. Upon receiving the request, the SP looks up a local table that is indexed by D to find the hidden path associated with D (the SP may support multiple services simultaneously). The SP verifies if the request is within the rate limit; if the number of requests forwarded to D has already reached the rate limit, the SP simply drops the request, otherwise it encapsulates the request in an outer header with the SP's unicast address ($SP_a^r$ in the figure) in the source address field and the hidden path identifier in the destination field and sends it to D. The network forwards the request to the destination using the hidden path identifier. Note that we use multicast addresses as hidden path identifiers.

When the destination receives the request, it decides whether it will accept S (Sec-

tion 4.2.4.2). If not, D can quietly drop the request, or as an optimization notify the SP to block future requests from that source. If the request is accepted, D chooses a DP and constructs a hidden path to it, if such a path does not already exist. D then generates an authorization token using the information in the request to identify S and creates an encapsulated response that has the DP's address and the authorization token for S. The response additionally has a rate limit for the amount of traffic S can send to D via the DP. The destination encrypts the response and sends it via the same SP from which it received the request. As shown in Figure 4.1.3, the encapsulated packet's destination address is the SP's real address $SP_a^r$, and the source field is filled with the hidden path identifier $HP_{ad}$. This way D does not have to reveal its IP address and any error messages (such as ICMP) generated inside the network can still be forwarded to D using the hidden path. Upon receiving the response packet, the SP decapsulates it and forwards it to the source using its own anycast address ($SP_a$). Note that the SP does not need to reveal its unicast address to the source since this is a stateless exchange.

After sending the response through the SP, the destination also sends a similar encapsulated response to the selected DP notifying it about the acceptance of S. However, the response sent to the DP can be decrypted by the DP as well. The DP decapsulates the response, creates a local table indexed by D that has the authorization token for S, the hidden path identifier to reach D, and the rate limit specified by D. The DP forwards this second response to the source.

From the responses the source learns about the DP, the authorization token, and the rate limit and is now ready to send/receive data to/from the destination. If S does not receive a response it can retry the request multiple times. Note that S may not receive the response due to several reasons: the rate limit at SP may have reached, the request may be lost in the network, the destination may have rejected the source, or the response from the destination is lost.

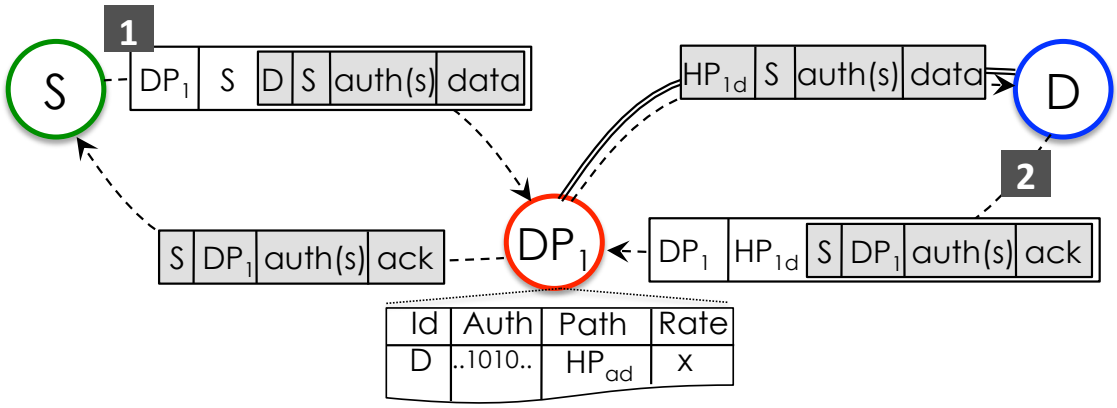


**Figure 4.1.4:** Data Transfer phase

#### 4.1.3.3 Data Transfer

Figure 4.1.4 shows the data transfer phase. S encapsulates the Epiphany data packet and sends it to the DP ($DP_1$ in the figure). The data packet has the authorization token given to S. Upon receiving the data packet, the DP decapsulates it, verifies S using the authorization token and checks if the data packet is within the rate limit set by the destination. If the checks satisfy the DP looks up the hidden path identifier to reach D, replaces the destination address field in the packet with the hidden path identifier ($HP_{1d}$), and forward the data packet to the destination. If on the other hand, the verification fails or if the packet has reached the rate limit, the DP will drop the packet. When the destination receives the packet it too verifies the authorization token from S and checks if the rate limit assigned to S is satisfied. If the verification succeeds, D sends acknowledgments or data packets back to the source via the DP. Figure 4.1.4 shows the acknowledgement messages from D to S, which go through the DP.

### 4.1.4 Threat Model

The problem we are addressing in this thesis is denial of service attacks from a distributed attacker against a critical service. The attacker may perpetrate the attack in many different ways, by attacking the application/protocol weaknesses or by target-

ing the network infrastructure that provides the service. We do not address attacks on application weaknesses since we assume they may be dealt by hardening the application software, but rather we focus on preventing infrastructure level attacks, specifically flooding attacks originating from a distributed attacker. We assume the attacker is attempting to disrupt the service to an unknown set of clients. Note that if the attacker knows the specific clients it wishes to deny service to, the attacker may directly flood them or target their surrounding infrastructure to deny service to those clients.

Since Epiphany hides the service address, the attacker may not be able to apply directed attacks on the service, but may target the Epiphany infrastructure, specifically the proxies, discovery service and intermediate routers to deny service to legitimate clients. We assume the attacker is aware of the destination's strategies and any defenses placed to counter the attacks. Some proxies may become compromised or may misbehave due to software failures. Moreover, the attacker may employ source address spoofing on these compromised proxies to masquerade as some other nodes to avoid detection. We assume some routers in the network may be compromised, but that the attacker cannot arrange for specific routers to be compromised. That is, the attacker may able to gain access to a few routers in the network, but does not have the ability to compromise specific routers on a certain network path.

#### 4.1.4.1 Attacks Against Proxies

The attacker may target the SPs and DPs to deplete their resources and cause service disruptions. First, since the SPs have public addresses that are know to everyone in the network. The attacker may resort to brute-force flooding the SPs with large volume of traffic to consume bandwidth resources on the network paths to the SPs. The flooding causes legitimate requests to be dropped at congested routers. Another more effective attack is when the attacker sends a flood of valid requests to consume the requests tokens at SPs. In Epiphany, the SPs do not authenticate requests (they do not make decisions

about which sources are malicious or legitimate). So if $R$ is the request rate limit at an SP, and $r_r$, $r_l$ are the attacker and legitimate client's combined request rates, then legitimate clients on average will get $O(\frac{r_l}{r_r+r_l})$ of the $R$ request capacity. But, if the attacker's rate is much higher, i.e., $r_r >> r_l$, then legitimate clients may fail to reach the destination in some expected duration (the duration is not more than few minutes). This type of attack is well studied in capabilities [7, 34, 48], but here we refer to it as *Denial of Connection* (DoC) attack.

Second, unlike the SPs, the DPs are not exposed to everyone in the network but rather a source must go through an approval process to learn about a DP. However, in an open service model in which anyone can request service from the destination, the destination may not have sufficient information about certain clients to readily decide if they should be approved or not; for instance, a new client that is requesting access to the destination for the first time. An aggressive policy at the destination might deny access to such clients, while a more open policy may approve them. Some of these clients could be malicious or part of the attacker botnet. Moreover, a previously approved client may have become compromised. Under these circumstances, the DP is now known to the attacker. The attacker may attempt to flood the destination through the DP, but the DP will filter out most of the excess traffic due to the rate limit set by the destination. However, the same DP may be used by multiple clients to reach the destination. The attacker can flood the DP to consume its resources (or the resources on the paths to the DP) and as a result, legitimate clients using that DP will experience a denial of service or serious degradation of the service performance. In other words, the attacker causes collateral damage to legitimate clients by targeting the DPs. The attacker may also game the decision system at the destination into revealing multiple DPs in order to make the attack more effective.

#### 4.1.4.2 Attacks from Compromised Proxies

The attacker may compromise the SPs and DPs, or the proxies may become malicious and collude with the attacker. A compromised proxy may not conform to the rate limits set by the destination, or may attempt to recover critical information about the clients and/or attempt to obtain the authorizations given to the clients. Through these compromised proxies, the attacker may funnel large volume of traffic to flood the destination. The attacker may also vary the attack traffic and/or the attacking proxies to avoid detection. As a result of these attacks, not only do the clients using the compromised proxies get affected, but the flood may cause collateral damage to legitimate traffic from other proxies.

#### 4.1.4.3 Attacks on Hidden Paths

Since hidden paths are a critical component in Epiphany, the attacker may attempt to hijack the hidden paths or may trick the destination into creating more hidden paths in order to consume its resources. In Epiphany, we do not consider penetration attacks [41], which are possible in other location hiding architectures such as SOS and TOR. In these attacks, the attacker compromises a series of nodes along the path towards the destination to learn about its IP address. However, to achieve this attack in Epiphany, the attacker must first compromise a proxy and subsequently compromise every router on the path to the destination. Ju Wang et al. [41] argue that making the intermediate path longer and employing heterogenous mix of operating system software on intermediate nodes makes penetration attacks much more difficult. In Epiphany, the proxies could be located farther away from the destination, and moreover Internet routers run different software compared to hosts and they are generally well managed. Therefore we assume these type of attacks are not possible in Epiphany.

#### 4.1.4.4 Attacks on Discovery Service

The availability of proxy directory service is essential for the clients to reach the service. Without it, clients have no way to know which proxies to use or what proxies are available at a given time. The attacker may target the directory service, in this case the DNS, to prevent clients from learning about the SPs. Although defending DNS is outside the scope of this work, we discuss several well known techniques to make proxy discovery through DNS robust to such attacks in later sections.

## 4.2 Epiphany Components: Proxies & Destination

We now describe some of the Epiphany components in greater detail including proxy discovery, SPs, DPs, the destination and its network.

### 4.2.1 Proxy Discovery Using DNS

In order to send requests to the destination the source must learn about at least one SP's address. We refer to this process as *proxy discovery* and achieve it using DNS. Assume that the source S has a service name it is attempting to reach. The source could have learned this name through some external means: an email, a web page (URL), or word of mouth. In Epiphany, the service name is synonymous with D, since the destination does not have a public IP address. To reach the destination, the source contacts the DNS and sends a query for the service name asking it for the A (or AAAA) records. The response contains IP addresses of nearby anycast SPs (if multiple anycast SP groups are involved) that the source could use to reach the destination. Epiphany relies on DNS for proxy discover mainly because it is widely available and moreover techniques to make DNS robust against attacks can be employed to make the proxy discovery robust. For instance, multiple redundant name servers could be employed for better availability and the name servers themselves could use anycast to separate the attacker regions.

Moreover, the DNS entries for SPs can now have longer time-to-live (TTL) values to increase availability through caching. That is, clients which have already consulted the DNS and learned about the SP address will be able to cache the result for long periods. Even if the DNS is temporarily unavailable due to an ongoing attack, the clients have the cached SP address to reach the service.

A key requirement in Epiphany is that the destination be able to add/remove SPs quickly and frequently to adapt to varying conditions, such as increase in load or in response to attacks on the SPs. But, large scale updates and quick propagation of changes is difficult to achieve in the DNS. However with anycast, a large number of SPs map to one or a small number of addresses in DNS; hosts acting as SPs can change frequently, but the DNS records can remain unchanged for long periods (i.e., the TTL values can be set to a longer duration). Note that if the destination must change the DNS records, we assume it has a secure path to the DNS servers to update the records, or this task may be delegated to a third party.

### 4.2.2 Setup Proxies

The setup proxies are the primary contact point for a source to reach the service and hence must have public addresses. However, the attacker may target the SPs to deny service to legitimate clients. Approaches such as SOS [17] and Mayday [5] employ a large number of proxies and use DNS tricks to evade the attacker (for instance, they suggest giving different SPs to different clients based on the client's location). Moreover, these approaches require a client to authenticate itself to a proxy to send a request. Unlike these, Epiphany has an open service model in which any client can send a request through the SP. However, to prevent indirect flooding the destination sets limits on the number of requests each SP is allowed to forward. Unfortunately, as we described earlier, this makes the attack easier, because the attacker only needs to send sufficient requests to consume the entire request token capacity at the SPs, thus denying legitimate

clients from being able to reach the service. Playing DNS tricks may not help, since the attacker may enumerate all SPs using the distributed botnet and consume the request capacity at all SPs. Increasing the number of SPs may not help either, as the combined rate from all SPs has to remain under a threshold to prevent flooding the destination with requests. The literature has proposals based on resource proofs to guarantee fairness to legitimate clients under these conditions. For instance, PortCullis [34] suggests using cryptographic puzzles inside request packets. However, Portcullis requires making fundamental changes to routers (both hardware and software) in order to support cryptographic puzzles. Our goal in Epiphany is to operate within the current Internet infrastructure and make fewer changes to Internet components for rapid deployment.

We first outline a few important properties for the SPs and discuss our solution to combat DoC attacks in more detail next:

- *Open Access*: In Epiphany, any source can send a request through an SP. The SP does not authenticate clients. Such a model allows the SPs to be lightweight and easy to deploy, since there is no user authentication information to store at the SPs.

- *Connectionless*: A source need not establish a connection with the SP to send requests, it simply sends a request and the SP forwards it to the destination. Connectionless setup allows the SPs to change without causing service disruptions. If an SP is removed, clients will send their requests to a different SP.

- *Regionalized*: Using IP anycast, the SPs form distinct regions in the network. Requests generated in a region are automatically directed by the underlying routing system to the nearest SP in that region. Therefore bots in one region cannot flood SPs in a different region. In other words, regionalizing helps disperse bots and free clean networks from unclean networks. Anycast has been shown to be effective for protecting DNS root servers [23, 25].

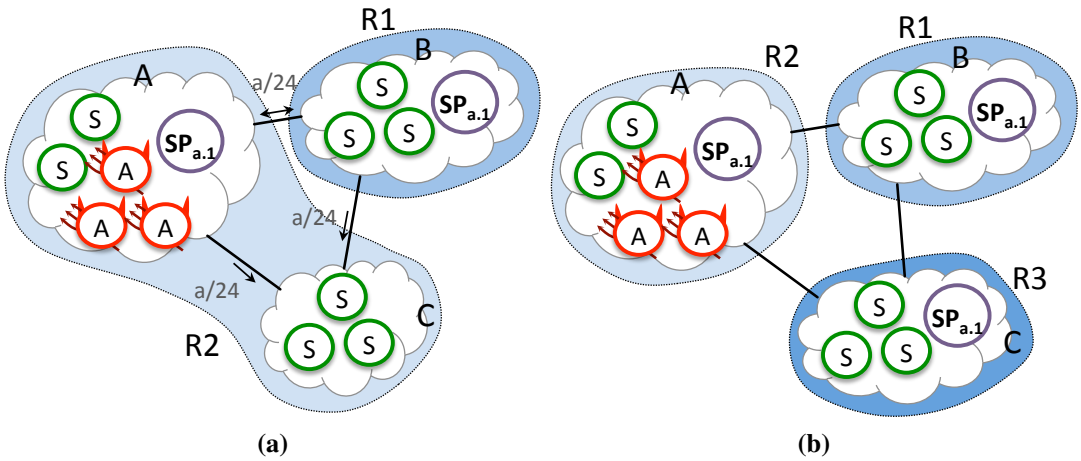


**Figure 4.2.1:** (a) Anycast SPs form distinct network regions, requests generated in a region are routed to a local SP. (b) Attackers are further confined to smaller regions by instantiating newer SPs.

### 4.2.3 Localizing Attackers Using Anycast SPs

To combat DoC attacks we propose to use Anycast SPs. RFC 4786 [4] describes anycast as "the practice of making a Service Address available to a routing system at Anycast Nodes in two or more discrete locations", i.e., in the anycast model a single IP address can have multiple points of presence in the Internet. We believe that although bots may be present everywhere, some networks however small they may be are cleaner than others (we borrow the definition of clean and unclean networks from [35]), i.e., have very few or no bots, but attackers in unclean networks can interfere with clean networks if all SPs are reachable from any network. It is these clean networks we wish to liberate from attackers. Using anycast, it is possible to restrict the accessibility of an SP to within a small network region. Multiple anycast node instances form distinct network regions. A request generated by clients in a region is automatically directed by the underlying routing system to the nearest anycast SP in that region, thus attackers are restricted to only targeting the SP in their own regions. If a region does not have attackers, legitimate clients in that region are unaffected. Figure 4.2.1a shows a simple

example of anycast SPs. Networks that host the SPs announce a route for the SP to neighboring networks. These networks, if not hosting an SP themselves, accept the best possible route (depending on the routing protocol in use) from the neighbors. In Figure 4.2.1a networks $A$ and $B$ announce a route for prefix `a/24` to network $C$, which then accepts the route from network $A$. When clients in network $C$ send requests, the network forwards them to the anycast SP in $A$. Attackers in $A$ can only flood the SP in $A$ but cannot prevent clients in $B$ from reaching the destination. On the other hand, clients in $C$ may not be able to reach the destination since they are using the SP in $A$, which is under attack. Note that the routes for anycast SPs may be locally scoped or global as defined in [4], i.e., a route for an anycast SP is either limited to a small network/subnet, or announced to outside networks. For instance, an Autonomous System (AS) may have multiple anycast SPs, some may be locally scoped so that the clients inside the AS can use them, while some anycast SPs may be announced outside the AS for neighboring networks to use. Scoping the anycast announcements gives more control to the network operator on who is allowed to use an SP, and also confines failures due to attackers to a small region.

Now, suppose the destination is able to identify a region with both attackers and legitimate clients, say perhaps through an external entity, by monitoring the load on SPs, or through explicit signaling from SPs. The destination can then instantiate more SPs in that region to further split it into smaller regions. In the example in Figure 4.2.1, although $C$ has no attackers in its own network, clients in $C$ may fail to reach the destination due to the attackers in $A$. Assuming the destination has access to a potential SP host in $C$, it can instantiate a new SP in $C$ as shown in Figure 4.2.1b. For instance, the destination could ask its providers for a new SP, it could reserve some SPs for later use, or the affected network may provide D with a new SP. In Figure 4.2.1b, after adding a new SP in $C$, the clients in $C$ are no longer affected by the attackers in $A$, or in other

words attacks are now localized to within the network $A$. To further dilute attackers, the destination could add more SPs to a region to split it into more regions. Region splitting can be applied all the way down to subnet level, at which point, clean subnets become separated from unclean ones. We believe this sort of localization is a first step to combating DDoS attacks. Bot activity becomes more visible and the attackers now become a local problem for operators in unclean networks. If these networks/regions do not take necessary measures, D can tell the SPs to block all requests from those regions. For instance, assuming the destination is able to identify bots, it may notice that requests from an SP all belong to bots and inform the SP to disable accepting requests by setting its request rate limit to zero. The bots in the region are now rendered ineffective.

### 4.2.4 Further Limiting DoC Attacks

As a result of localization, some networks may locate and disable the bots in their networks, but in some cases this may not possible or easy, i.e., a region may end up with both attackers and legitimate clients. For instance, a bot may be a DSL customer paying for connectivity. The DSL provider cannot easily remove this bot without risking losing the customer. Another issue is that a region cannot be made smaller due to unavailability of SP hosts. Under these conditions, limiting the attackers using additional mechanisms would be helpful to increase the chances for legitimate clients to reach the destination. One approach is to filter requests based on their source IP addresses at the SPs. For example, if the SP has sent a request for a particular host in the last $x$ seconds, then further requests from that source (to the same destination) are ignored. However, a bot may spoof its IP address and masquerade as other legitimate hosts in its local network and deplete the request resources at the SP. We sketch out two designs inspired from existing resource proof based approaches [34, 40, 43] to limit local attackers in these regions. An SP may implement any of them, but we assume clients can become aware of which mechanism is in use.

The first approach is similar in spirit to Speak-Up [40]. Earlier we noted that legitimate clients on average get $O(\frac{r_l}{r_r+r_l})$ of the $R$ request resources at an SP. If we make the clients send requests at the same rate as attackers i.e., $r_r \approx r_l$, then there is a 50% chance that one of them will get through to the destination. More specifically, when a source S is unable to reach the destination in an expected amount of time, it starts sending requests at a maximum rate allowed by its uplink capacity, i.e., the source becomes a request flooder for a short period of time (until it receives a response from the destination). Even if the attacker's aggregate rate is higher than a single client (due to the presence of multiple bots), the client still has more chance of success than when requesting at low rates. The problem with this approach however is that it may cause congestion in the network due to the increased traffic from all clients. However, since the anycast SPs restrict the requests to a small network region, we believe the congestion will not propagate to deeper parts of the network where it can adversely affect the network performance of a large number of clients.

The second approach is to employ cryptographic puzzles similar to [34,43,44]. However, our approach, although inspired from these proposals, is different in one important aspect – only the SPs verify puzzle solutions. We do not involve intermediate routers between the SPs and the clients in puzzle verification. This has the benefit that routers do not need hardware or software changes and thus makes deployment easier. In this approach, a client S starts by sending a request to the SP. If the SP is unable to forward the request, it returns a nonce $N_s$ to the client asking it to solve a cryptographic puzzle. The client chooses another nonce $N_c$ and a puzzle difficulty level $l$ and computes a solution $x$ such that the last $l$ bits of $P = H(x, N_c, N_s, l)$ are all zeros. Here $H()$ is a universally known hash function. $N_s$ guarantees puzzle freshness, i.e., a client cannot precompute puzzle solutions since it does not have $N_s$ before contacting the SP. The SP issues different $N_s$ to different clients and keeps track of them by entering $N_s$ in

a bloom filter (for easy verification). The nonce $N_c$ distinguishes the puzzle solutions from different clients. Finding $x$ requires the client to brute-force search $2^l$ solution space, which takes a random amount of time depending the puzzle difficulty level $l$. For instance, with $l = 2$, the client needs to search 4 values of $x$ to find a solution. Once the client finds a solution, it sends a second request with the parameters $P$, $N_c$, $N_s$, $l$ and the solution $x$. The SP can easily verify the solution $x$ by matching $P$ sent by the client with $P' = H(x, N_c, N_s, l)$ computed by the SP. Note that the SP also verifies $N_s$ using the bloom filter. The SP records the solution $x$ along with the solution parameters so that S cannot reuse the solution in subsequent requests. If the puzzle solution verifies, the SP checks if the puzzle solution is above a threshold limit for that time period (can be a second) and forwards the request to the destination. If however, the puzzle solution does not verify or if the difficulty level is not above the threshold limit, the SP drops the request forcing the client to retry. The client will have to retry with a greater puzzle difficulty level than before to increase its chance of success.

Using this approach, the attacker must either compute unique puzzle solutions to sustain an attack rate or resort to brute-force bandwidth exhaustion attacks. As we noted earlier, the localization effect of the SPs dilutes the attacker bots. Hence, if the SP is more provisioned than the combined bot resources in that local network, the attacker will not succeed with brute-force attacks. On the other hand, the attacker may use the bots to solve puzzle solutions to sustain a constant attack rate. Using this, the attacker may not prevent legitimate clients from reaching the service, but may delay them long enough for the users to give up on the service. We evaluate this behavior in Chapter 5.

#### 4.2.4.1 Assigning Rate Limits

The primary reason for rate-limiting is so that if an attacker floods the SP, the flood is throttled at the SP rather than directed at the destination. However, rate limiting also affects how successful a client will be in reaching the destination. We outline a few

techniques for assigning rate limits to SPs.

Suppose the destination sets aside $C_r$ of it available bandwidth for requests. D could assign this capacity to the SPs in several ways: 1) it may assign equal rate limits to all SPs, i.e., if $|N_{sp}|$ is number of SPs, then each SP gets $\frac{C_r}{|N_{sp}|}$ of the capacity as the rate-limit, or 2) D may assign different rate limits to different SPs, but the combined rate adds up to $C_r$, or 3) D may adapt the rate limit based on demand. In the first case, regardless of the client population or demand, each SP is only allowed to forward a fixed number of requests even if some SPs have no requests to forward. In this situation, some clients may not be able to reach the destination if their SPs are heavily loaded, while some other SPs have unused capacity. The second approach might address these issues, but is much harder to achieve if the destination does not know ahead of time how the clients are distributed or how the load will be distributed. The third approach allows the destination to dynamically change the rate-limits depending on demand, but requires addition processing overhead at the destination.

#### 4.2.4.2 Authorizing a Source

In Epiphany, we assume the destination has policies to decide which source to authorize. These could be based on local policies at the destination, prior authorizations given to important clients, reputation systems, blacklists of misbehaving sources and so forth. For example, the destination may never authorize a source that sends a malformed request or a source found in a blacklist. Reputation systems such as the one described in [26], could be used to distinguish important clients from new clients that the destination has never seen. Once the destination decides to authorize a source, it generates an authorization token and selects a DP to return to the source. The authorization token is simply a random nonce given to both the source and the DP. However, the problem is that if the destination sends this information in the clear, a compromised SP or an intermediate node may be able to learn about the DP and steal the authorization issued

to the source. To address this, we assume the source, destination and DP exchange cryptographic keys to encrypt the future packets.

We roughly sketch an example of one such mechanism, but note that different applications may employ different mechanisms. Suppose a source S has a signed reputation value and the public key of the destination D (learned from perhaps a previous interaction, or may be distributed to S using some other means). The source starts out by picking a new shared session key, and encrypts the key and the reputation value using the public key of D. The source sends this information in a request packet. Only the destination can decrypt and learn about the session key since it has the private key. The destination will verify the reputation value and may generate a new signed reputation value for S. Subsequently, the destination generates a response packet with the new reputation value, the authorization token, the DP's address, and the rate limit. However, D encrypts the response using the session key from S and sends it via the SP. Since the SP does not have the session key it cannot decrypt the response to learn about the DP or the authorization token. Now, to notify the DP the destination generates another response with S's IP address, the session key from S, the authorization token, the new reputation value, and the rate limit and sends it to the DP. However, the destination encrypts this response using the DP's public key (learned during initial configuration). The DP decrypts the response using its private key, learns about S, the authorization token and the session key. For every future packet that S sends, it includes the authorization token and encrypts the data packet using the session key. The DP has the session key to decrypt the data packet from S and verify it using the authorization token. The DP simply matches its version of the authorization token with the token included in the data packet. The DP forwards the encrypted data packet to the destination. The destination may periodically renew the authorization token given to S.

#### 4.2.4.3 Assigning DPs

Inaccuracies in the decision process at the destination may cause it to incorrectly authorize an attacker. For instance, a default policy at the destination might be to accept any new client, but without sufficient knowledge, the destination may accept a new host that belongs to the attacker's botnet. The attacker may not be able to flood the destination through the DP using this bot (since the DP filters out excess traffic), but may still cause collateral damage to some other legitimate clients using that DP.

To address this issue, the destination may assign DPs based on different trust tiers of clients. In this model, lower tier DPs are assigned to new and untrusted clients, while higher tier DPs are reserved for trusted clients or those with a history of good behavior. While a determined attacker can still enumerate all the lower tier DPs, this will only affect new untrusted clients. The service can protect legitimate clients by having a pool of low trust DPs that are expendable. For example, suppose the destination knows the properties of a DP such as its current load, bandwidth, or trust level, it could assign better DPs (i.e., DPs with fewer clients, DPs not heavily loaded, trusted DPs) to more important clients while assign a fixed set of DPs to untrusted clients. This way, important clients will be unaffected since they are on different DPs. Some new clients that are legitimate will fail to reach the destination, but they can restart the setup process to receive a different DP. As the destination builds more trust or reputation about a source, it can progressively move that source to better DPs in future interactions. In addition, the service may distribute the DPs far from each other, where there is less likelihood that an attack on one DP will affect others.

### 4.2.5 Requirements for the Destination Network

The strength of location hiding architectures depends on how well the destination is kept hidden. Otherwise, secondary defenses such as filtering rings may be necessary [13,17].

To achieve strong location hiding not only does the destination software needs to be robust against accidental information leaks, but the destination network itself must have a few important properties. Note that the properties we list below are not mandatory, but would enhance protection for critical services.

For maximum protection the service can be placed in a separate network that does not have any public components. To see why, consider a hidden web-server such as www.private.fbi.gov. If the web-server is co-located with the rest of the public fbi.gov domain, the attacker could simply target any host in fbi.gov domain to indirectly inflict damage. Another possibility is to locate the web-server is a completely different domain. For example, locate the www.private.fbi.gov web-server in google.com domain. Though this might make it harder for the attacker to guess the web-server's host domain, public components if any present in the host domain might give away the hidden server if they are compromised (for instance, they may discover the hidden server through local network scans).

Another interesting step is to completely disable routing announcements *for* the hidden network into global routing space (but the routing announcements from other networks are still received by the hidden network). Since there are no public components in this network, and since all communication with the destination happens explicitly through hidden paths, the destination IP prefix is not needed to route any packets to the destination. The access routers to the hidden network are the only ones that know about this network. If remote management of the service is required, the destination could have a permanent hidden path to a special remote management proxy. This allows the operators to reach the destination without having to announce routes to the global routing space.

#### 4.2.5.1 Handling Misbehaving Proxies

The attacker could use compromised proxies to flood the destination. Although the destination may know that it is under attack, it may not know which proxy is causing the flood if compromised proxies employ source address spoofing. To address this, we develop a path identifier approach similar to Pi [45] in Section 4.3.5. Briefly, this approach gives an identifiable signature to all packets originating from the same proxy. The destination can match the signatures and remove the hidden paths to misbehaving proxies. We discuss the mechanism in more detail in Section 4.3.5.

Compromised SPs may attempt to steal the authorizations given to legitimate clients, may block certain legitimate clients from reaching the destination or worse, may masquerade as a legitimate source and try to obtain authorization from the destination. In the first case, we discussed a mechanism by which a client could exchange cryptographic keys with the destination to encrypt the response packet. The SP cannot learn about the authorization token since it does not have the keys to decrypt the message. If a compromised SP drops requests from a particular client, the client could retry the request through a different anycast SP (if multiple anycast SPs are available). In the last case, we noted that a legitimate client may include a prior authorization token or a digital certificate in the request to identify itself to the destination. However, if neither of these are available, a new source is limited to sending a simple request with its IP address and a session key encrypted with the destination's public key. A compromised SP could become man-in-the-middle attacker and easily replace the session key with its own key and steal the authorization (and DP address) given to this new source. However, we note that this limitation is not unique to Epiphany alone, but is possible in other defenses such as capability architectures. A compromised SP may also send a simple request posing as a legitimate client (i.e., without including a prior authorization or digital certificate). However to combat attacks, our tiered approach to assigning DPs can be used. When

the actual legitimate source sends a subsequent request, it will include the prior authorization token or digital certificate and may receive a different higher tiered DP. Finally, similar to the compromised SPs, compromised DPs could also block certain clients from reaching the destination or may steal authorizations given to legitimate clients and given them to the attackers. The attackers may use these to send bogus traffic to the destination. However, once the destination detects this behavior, it can move the legitimate clients to a different DP. Also, we assume the client may exchange additional session keys with the destination to protect its traffic from sniffing attacks.

## 4.3 Epiphany Components: Hidden paths

Hidden paths are a fundamental component of the Epiphany architecture. Unlike unicast routing where knowing the destination's address is sufficient to send traffic to it, hidden paths do not reveal the location of the destination or allow anyone to send traffic to it unless two conditions are met. First, a node must know about a hidden path address that uniquely identifies a forwarding table entry at each intermediate hop on the path, and second, the node must have a forwarding table entry pointing to the correct next hop node (out of the possible set of neighboring nodes) on the path to the destination. We call our paths hidden for this very reason, i.e., each router on the path only knows the next router on the path, but do not where the path originated from or where it may lead to.

Hidden paths allow the destination to control how the proxies send traffic. The proxies may become compromised or turn malicious and may be used to attack the destination. Under these conditions, cutting them off from sending any further traffic is necessary. In Epiphany, the hidden path is created between the destination and a proxy (either SP or DP). We assume all routers between the proxies and the destination are Epiphany aware (we relax this requirement and discuss alternatives to work around

non-Epiphany aware routers in Section 6.1). To successfully hide the service, we now describe a set of goals/properties we wish to achieve for the hidden paths. The actual mechanism to construct the hidden paths is discussed later.

### 4.3.1 Properties of Hidden Paths

- *Strong location hiding*. Knowing the hidden path identifier does not divulge the destination's IP address or its location either implicitly or explicitly. In Epiphany, the hidden path is an address $G$ that the proxy stamps on packets in order to send them to the destination. This address is different from the destination address, and knowing it does not give away any information about the destination.

- *Cannot be easily forged*. Constructing hidden paths requires secured message exchange between the destination, intermediate routers, and a proxy. All the state regarding the hidden path is maintained inside the network and is tied to the address $G$ and a secret shared between the destination and intermediate routers. An adversary cannot easily forge this hidden path to inject packets towards the destination.

- *Path may expire or can be removed*. The hidden path is created dynamically upon the destination's request, and expires after a time interval. The path is periodically refreshed to keep it active. Hidden paths may also be explicitly removed upon the destination's request.

- *Efficient*. A router does not incur any additional overhead when forwarding an Epiphany packet on a hidden path compared to forwarding a unicast packet.

- *Compatibility with IP packet structure*. Hidden paths do not change the IP packet structure and are compatible with the current IP packet forwarding mechanisms.

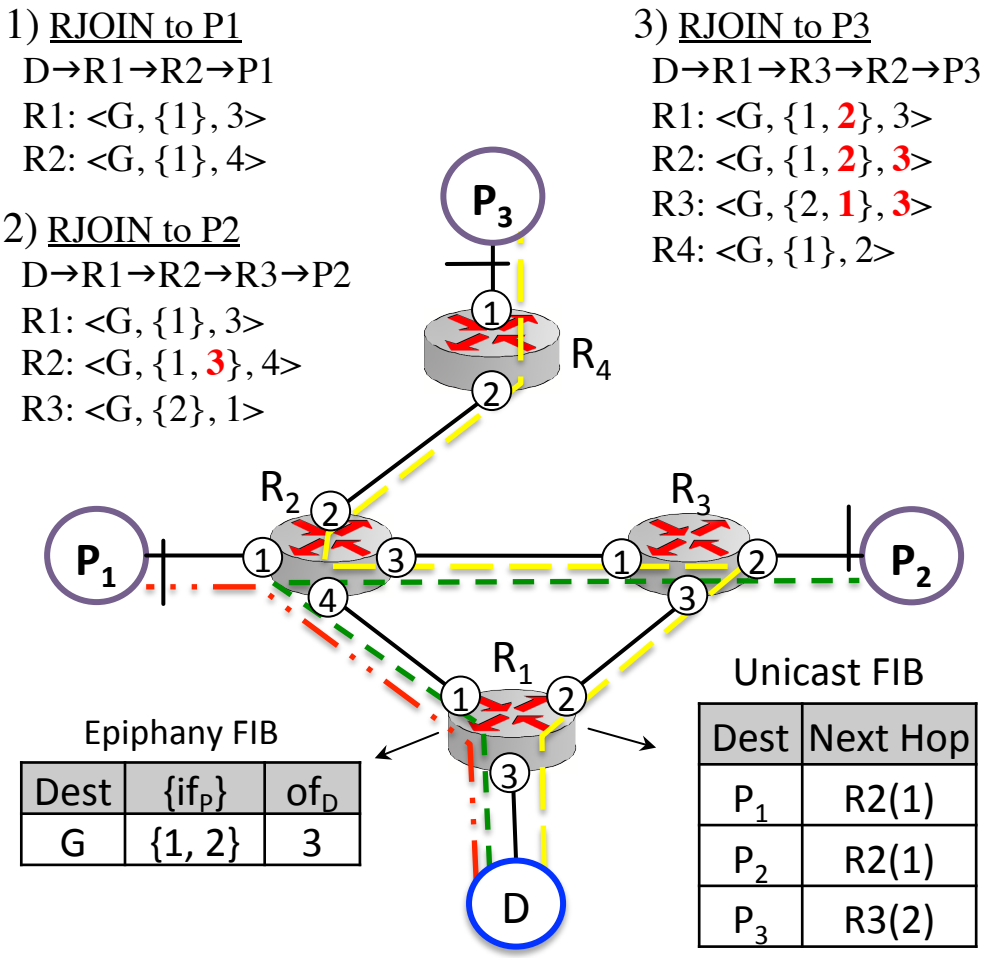


**Figure 4.3.1:** Example showing the construction of hidden paths

This is an important requirement for rapid deployment unlike capability architectures that change the IP packet structure and router architecture.

### 4.3.2 Constructing Hidden Paths using Reverse-Multicast

Our implementation of hidden paths shares many similarities with multicast routing. In multicast, a set of receivers express interest to receiver traffic addressed to a group address (also known as multicast address), while a sender simply sends traffic to this address. Each router starting from the one closest to each of the receiver builds a multicast forwarding table entry towards the source. The forwarding table entry has the

group address, the interface on which the router will receive packets addressed to the group address, and the interfaces on which it should send these packets towards the receivers. However, in this model, the source does not know the number of receivers or their corresponding addresses. In addition, knowing the group address does not divulge the location of the receivers, since to know that, one has to look at the forwarding tables of each intermediate router on the path(s) from the source to the receivers. Each router only has partial information about the path, i.e., the previous and next hops on the path between the source and the receivers.

Unlike multicast, in Epiphany, we want a *group of senders* to send traffic towards a *single receiver* whose address they do not know. We refer to our implementation as *reverse-multicast* due to its similarities with multicast. For illustrative purposes, we refer to the example shown in Figure 4.3.1 to describe the implementation. The router functions in pseudocode are outlined in Algorithm 1.

When the destination decides to build a hidden path to a proxy $P_1$, it picks a random IP address $G$ from a range of addresses. For security, the address range is chosen to be large to avoid address collisions (different hidden destinations may randomly pick the same address) and it cannot be used for unicast routing to keep forwarding on hidden paths separate from unicast paths[1]. The destination lookups proxy $P_1$ in its unicast forwarding table and sends an *rjoin* message to the next hop router $R_1$ on the path to $P_1$. The format of this message is shown in Figure 4.3.2. The IP header has $R_1$ in the destination field and the source field has the address $G$.

When $R_1$ receives the rjoin, it looks up $P_1$ in the unicast forwarding table (*Unicast-FIB*) and identifies the interface $if_{pi}$ to reach $P_1$. Note that the unicast FIB is populated at router $R_1$ through conventional IP routing. If there is no entry for $G$ in $R_1$'s *Epiphany*

[1]To simplify address allocation, one may use multicast address space for this purpose.

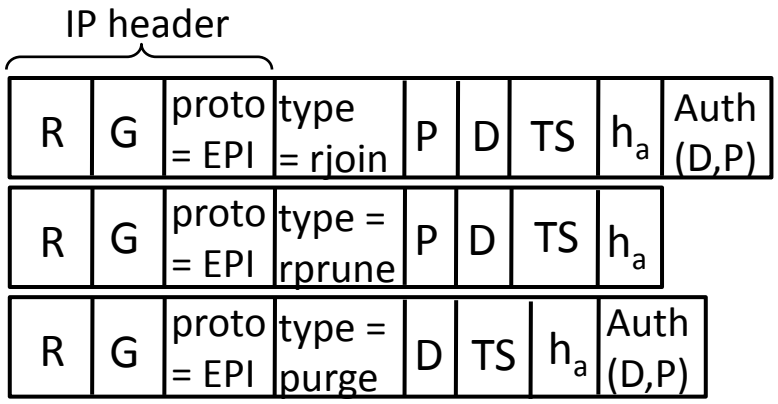


**Figure 4.3.2:** Epiphany message formats

*Forwarding Information Base (EFIB)* table, it creates a new entry for $G$ in the $EFIB$ with $if_{pi}$ as one of the input interfaces and $if_{join}$ as the output interface, i.e., the interface on which $R_1$ received the rjoin. Otherwise, $R_1$ adds $if_{pi}$ to the already existing set of input interfaces for $G$, and replaces the output interface $if_{out}$ with the new interface $if_{join}$. $R_1$ forwards the rjoin towards $P_1$. Subsequent routers on the path to $P_1$ perform similar operations and deliver the rjoin to $P_1$, while building the hidden path in their EFIBs. After receiving the rjoin, $P_1$ verifies it using the $Auth(D, P)$ field and registers an entry for D. The $Auth(D, P)$ field contains authentication tokens exchanged between D and $P_1$ during initial configuration. An adversary cannot establish hidden paths to this proxy since it does not have the authentication tokens to verify the rjoin. Note that, although we describe hidden path creation for point to point links, routers may be connected to several other routers on a shared medium (such as Ethernet). In such case, we assume routers have additional link level mechanisms to correctly resolve the input and output interfaces to their corresponding previous and next hop routers.

Figure 4.3.1 shows the unicast path from D to $P_1$, and the EFIB entries created at each of the routers in response to the rjoin to $P_1$ (step-1). The EFIB entries create the hidden path in the opposite direction of the unicast path, shown using a red line in the figure. When D wishes to create hidden paths to proxies $P_2$ and $P_3$, it uses the same

address $G$ and sends rjoins towards $P_2$ (step-2) and $P_3$ (step-3). For example, when D sends the rjoin for $P_2$, router $R_2$ appends a new input interface (iface 3) to the existing input interface set for $G$. On the other hand, the rjoin for $P_3$ crosses the previous path established for $P_2$ (Figure 4.3.1, the yellow path crosses the previously established green path between $R_2$ and $R_3$), i.e., one of the input interfaces at $R_2$ (iface 3) now becomes an output interface. $R_2$ removes this interface from the input interface set and replaces the previous output interface to point to $R_3$. Note that, due to the rjoin for $P_3$, portions of red and green paths are deleted and $P_1$, $P_2$ use portions of the yellow path to reach D.

Using this method of constructing hidden paths, the state required at each router is proportional to the number of hidden destinations (one $G$ per destination). While this may seem state intensive, we expect the number of hidden destinations to be small, often limited to critical services. Moreover, we expect a gradual deployment strategy with the most important services moving to Epiphany and increasing the router resources as more services need defense. For example, a current Internet router holds roughly 350K unicast forwarding entries. If we increase this capacity by 20% and allocate it towards Epiphany, the router can support up to 70K hidden destinations. Note that only the routers between the proxies and the destination need to support this increase in forwarding state (since the hidden paths are only constructed on top of them), and moreover no inter-domain routing protocol is needed, which is a major stumbling block with IP multicast. We believe our proposed approach is a reasonable tradeoff to achieve efficient packet forwarding while keeping compatibility with the IP packet structure.

### 4.3.3 Handling Loops and Failures

Under stable unicast routing conditions, Epiphany creates loop free hidden paths. We prove this by contradiction. Suppose the destination constructs hidden paths to proxies $P_1$ through $P_{n-1}$ and the resulting paths does not have loops (Figure 4.3.3, loop free paths are shown in black). Now suppose the destination constructs a hidden path to $P_n$,

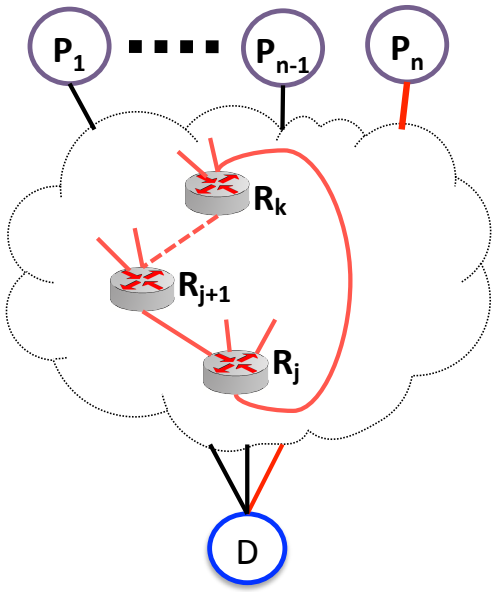


**Figure 4.3.3:** Forwarding loop in Epiphany hidden paths

but the resulting path has a forwarding loop, as shown Figure 4.3.3 with a red line. The only way this could occur is if one router's out interface points to the input interface of another router on the same path. For example, in Figure 4.3.3 some router $R_j$ points to another router $R_k$ on the path to $P_n$. However, since the rjoin replaces the outgoing interface at each router it traverses, router $R_j$ must have received an rjoin from $R_k$, and $R_k$ must have received it from $R_{j+1}$, which in turn must have received it from $R_j$. However, under stable routing conditions the rjoin for $P_n$ should have originated at the destination and traversed a loop free path to $P_n$, which contradicts the the loop formation in the forwarding state.

Although loops do not occur under stable routing conditions, they may be possible when the rjoin coincides with unicast routing changes. Moreover, network failures may cause the hidden paths to fail and as a result several proxies sharing a path segment associated with the failure or loop may not be able to reach the destination. Under these conditions, the Epiphany forwarding state at the affected routers must be removed. To achieve this, we use a soft-state approach – after updating the EFIB in response to an

rjoin, each router independently sets a timer for the input interface it just added (i.e., for $EFIB.G.\{if_{in}\}$). If no packets are received on this interface for the timeout duration, the interface is removed from the input interface set for $G$. When this set becomes empty, the entire EFIB entry for $G$ is removed. In order to keep the state active, proxies must send either data or *keepalive* messages. However, we additionally require that in order to continue sending data or keepalives, the proxies must receive periodic rate refresh messages (or data) from D. If no packets are received from D in the last $T_{rr}$ seconds (i.e., rate refresh interval), the proxies stop sending the keepalives (or data), which causes the router state to expire. Note that during a failure or a loop, D does not receive keepalives from certain proxies and it stops sending rate refreshes (or data) to those proxies. The proxies timeout after $T_{rr}$ seconds and stop sending keepalives. The state at the routers times out and will be removed. Eventually when unicast routing stabilizes, the destination can send rjoins to the affected proxies to create new hidden paths.

In Figure 4.3.1, assuming no loops or failures occur on the unicast paths, the soft-state approach removes unused state at routers. For example, after the rjoin for $P_3$, $R_2$ will switch its outgoing interface to 3, and $R_1$ will stop receiving traffic on input interface 1. This interface will no longer receive any keepalive messages, and after the timeout $R_1$ will remove it from the set of input interfaces for $G$.

### 4.3.4 Securing the Hidden Paths

If a different destination D' sends an rjoin for a $G$ used by a destination D, it may usurp the hidden paths created by D. This may happen unintentionally, as both D and D' randomly choose the same $G$, or D' may be an attacker who learned about $G$. In the first case, the probability of collisions can be reduced by choosing a large address space for $G$. The second case however, requires securing the hidden paths as they are constructed. In Epiphany, we adopt a simple solution: we make sure that the first destination to send

an rjoin for $G$ claims the EFIB entry for $G$ at a router until it decides to relinquish it at a later time.

Specifically, before the destination sends an rjoin for $G$ for the first time, it creates a hash chain $\{h_0, \ldots, h_n\}$ by repeatedly hashing a secret $k$ using a well known hash function $H$, i.e., $h_0 = H(k), h_1 = H(h_0), \ldots, h_n = H(h_{n-1})$. Sufficiently long hash chains can be generated offline for this purpose. The destination includes the last hash element $h_n$, called the hash anchor $h_a$ in the first rjoin it sends out. Routers associate the hash anchor $h_a$ with address $G$. For every future rjoin or control messages that the destination sends out, it includes the previous hash element to the one it recently used in the rjoin or control message. For example, D uses $h_{n-1}$ for the second rjoin it sends out. When a router receives the message, it hashes $h_{n-1}$ once to get $h_n = h_a$ and thus verifies the message. The router can now update the hash anchor by replacing the previous $h_a$ with $h_{n-1}$ to verify future messages. Note that not all routers receive every rjoin or control message that the destination sends out, and therefore they may have an older hash anchor. However, to verify a future message they simply have to repeatedly hash the new element to get $h_a$ (an index into the hash chain can be sent in the message to speed up this operation).

Now, assume the attacker learns about the hash element $h_n$ (possibly through a compromised proxy). However, given $h_n$ the attacker cannot easily generate $h_{n-1}$ due to the one way property of the hash function. Therefore the attacker cannot send rjoins or control messages with a new hash element in order to usurp the state associated with $G$ at routers. Although these type of attacks are not possible, the attacker may still game the system by flooding the network with bogus rjoin messages using the hash element $h_n$. As noted earlier, some routers have outdated hash anchors and may accept the bogus rjoin from the attacker and change the hidden path. As a result, traffic from some of the proxies will be directed to the attacker. However, this change in the hidden path

is only temporary, since the attacker does not have the original hash chain. When the destination realizes that the traffic is not arriving from some of the proxies, it will send rjoins towards the affected proxies with a new hash anchor and the hidden path will be updated. The destination may also proactively send the latest hash element to its proxies and tell them to send it in keepalives (an optional part of keepalive message). Routers can then update their anchor after verifying this new element. In the mean time, the attacker would have collected some traffic intended for the destination, but we assume the service uses application layer encryption to prevent the adversary from making use of this traffic.

The attacker may also target the hidden paths in two additional ways: 1) the attacker may send bogus rjoins/control messages with completely random hash elements, and 2) the attacker may send several rjoins to different $G$s to exhaust the forwarding state at routers. In both cases, the attacker will cause DoS attack on routers. In the first case, routers with an EFIB entry for $G$ will repeatedly hash the random hash element in an attempt to verify the bogus message. However, since the hash element is completely random, the router will take an indefinite amount of time to declare failure. The router can limit the number of rounds it will try before declaring failure, but the attacker can repeatedly send bogus messages to engage the router. The second case is similar, but instead of exhausting the processing resources of the router, the attacker could exhaust the state resources of the router. To limit this type of attack, routers could allocate a fixed amount of the EFIB resources to each of their neighbors. When they reach the limit for a particular neighbor, any future rjoins from that neighbor are ignored. The attacker may flood the network with rjoins, but will end up consuming the EFIB resources at the routers near his network; future rjoins will not propagate any farther. If the attacker just happens to share most of the paths with D, then D may not be able to establish hidden paths. However, since the attacker cannot precisely locate D, we believe the likelihood

of such an event is small. Moreover, we assume hidden networks and their surroundings are tightly controlled by the operators to prevent such attacks.

### 4.3.5 Identifying Misbehaving Proxies

Hidden paths give unrestricted access to the proxies to send traffic to the destination. If a proxy becomes malicious or gets compromised, the attacker may use it to flood the destination. In this case, the destination will know that there is a misbehaving proxy since it will receive packets at a rate much higher than the combined rate limits at all proxies. For example, if the rate limit at each proxy is 1 packet per second and if there are 10 proxies, then the destination should only receive 10 packets per second; If it receives more, it will know that there is a misbehaving proxy. However, the destination may not necessarily know which of the proxies is sending the traffic flood. A misbehaving proxy may spoof its IP address and masquerade the flood using different IP addresses. Worse yet, if the misbehaving proxy learns about other proxies in the network, it may spoof their IP addresses and fool the destination into blaming those proxies. Although this type of scenario may be avoided to a certain degree by employing source address filtering at the proxy access routers, we believe such a requirement is hard to achieve at all proxies. Moreover, the destination has no way to verify if a certain network hosting a proxy has indeed employed source address filtering.

Rather, we need a mechanism to identify packets arriving from different proxies without relying on their source IP addresses. To this end, we adopt a solution similar to *path identifier* (Pi) [45]. Pi inserts a path dependent signature into the *identification* field of an IP packet (IPv6 has a flow label field that could be adopted for similar purposes). Specifically, each router on the path from the source to the destination inserts a small portion of the signature. Therefore packets traversing the same path will have the same signature when received at the destination. However, Pi is somewhat limited in resolving the paths due to the limited space available in the IP identification field (16-bits); the

signature in Pi comes from the routers closest to the destination. With 2-bits per router (suggested value in Pi [45]), up to 8 routers closest to the destination can insert their signatures. In Epiphany, if a number of proxies share the same path segment, and if the segment is longer than the field length divided by the number of Pi bits per router (i.e., path length $> 8$), the proxies may end up with the same signature. The destination will have difficulty discerning the packets originating from these proxies. Instead, we want to capture some bits from the routers closer to the proxy, and some bits from the routers upstream of the destination. However, a misbehaving proxy may lie about the routers in its own network and insert bogus signatures into the IP identification field. For these reasons we need a slightly different algorithm than the one proposed in Pi [45].

When D sends an rjoin, it initializes a counter (i.e., sets it to 0) in the rjoin packet. Every router on the path increments this counter and sends back an *rjoin report* message with the IP addresses of the interfaces on which the rjoin is received and sent, the current counter value, and the Pi bits it will insert into the packet. The Pi bits are a 2-bit random nonce generated by the router (2-bits is the suggested parameter in the original Pi approach). The destination collects the the rjoin report messages from upstream routers and reconstructs a topology tree identifying the paths to proxies. The destination also has a list of path identifier/signatures at reach router. Note that a compromised proxy may generate bogus rjoin report messages for non-existent routers in its own network. If the proxy attempts to flood the destination using these bogus report messages, D will know about it and cutoff the proxy. For instance, D could set a limit on the total path length it will accept as valid. If it receives more report messages than this limit, D will cutoff the proxy.

Now that the destination has information about the upstream topology, it will decide which of the routers in the topology should mark the packets with Pi bits. Lets suppose that the destination is able to identify a router that has the same IP address prefix as the

proxy. Here, having the same prefix means that some intermediate router's IP address has the same initial $p$[2] bits as the address of the proxy. We denote this router as $R_s$. The destination could vary this prefix length to capture routers that are closest to the proxy (i.e., in the proxy's subnet) or routers that are farther out towards the border of the network hosting the proxy. However a misbehaving proxy may fabricate bogus routers near its network, i.e., router IP addresses with longer prefix $p$. If the destination chooses these routers, then detecting a misbehaving proxy may become difficult since the proxy will freely insert bogus signatures. One the other hand, if the destination chooses routers that are farther away from the proxy; proxies in the same domain will have the same signature. To address this, we propose to split the IP identification field into two fields; the first $m$ bits are set aside for routers closer to the proxy and the next $n$ bits for the routers upstream of D. To pick $R_s$, the destination chooses the first router that has the largest matching prefix on the path to each proxy. That is, matching the prefixes starting with the router closest to the destination, pick the first router that has the largest matching prefix (say a /16 or a /24) to a proxy. The router $R_s$ will insert its Pi bits in the $m$ bits and the routers upstream of it (towards the proxy) will insert their Pi bits into the remainder of $m$ bits, while the routers immediately upstream of D will insert their Pi bits into the $n$ bits of the IP identification field. To decrease the chance that bogus routers from the proxy are accepted as marking routers, the destination will use a smaller value for $m$ (say 2 hops).

After this step, the destination generates a list of indices one for each router on the path starting from each proxy, to indicate whether they should insert Pi bits (or not), and if so, where in the IP identification field they should insert those bits. Routers not selected to insert the Pi bits are given a 0 index. The destination sends this list of

---

[2]A prefix is typically denoted with the notation /$p$.

indices as a rate refresh message to the proxy. The proxy must forward this list, along with a counter that is initialized to 0, in a keepalive message to the destination. If the proxy lies about the information in the keepalive, D will know about it and cutoff that proxy (D must receive the keepalive with the original index list). When each router receives the keepalive message, it reads the counter, decrements it, and picks an index from the list corresponding to the counter. The router associates this index with the address $G$. When a packet from a proxy arrives at each router, the routers that have an index great than zero will insert their Pi bits into the IP identification field of the packet. If a misbehaving proxy attempts to flood the destination using spoofed addresses, the destination will be able to identify the proxy using the signature on the packets. Note that in this method, if some of the $m$ bits are from bogus routers inserted by a proxy, then D may receive packets with different signatures. However, the signatures will have the same $n$ bits and some portion of the $m$ bits (the proxy cannot arrange for this portion to be different). In this case, the destination may need to selectively cutoff proxies with the same matching portion of the signature. This way, D may not be able to readily find the exact misbehaving proxy, but the proxy will be eventually removed.

### 4.3.6 Removing Misbehaving Proxies

If the destination decides to remove the hidden path to a proxy, it stops sending rate refreshes causing the router state to expire. However, a misbehaving proxy may keep the router state active by disregarding the rules for keepalives. To remove such proxies, the destination sends an explicit *rprune* message towards the misbehaving proxy once it is identified. Figure 4.3.2 shows the format of this message, and Algorithm 1 describes the router actions. Routers upstream from the destination forward the rprune message until it reaches the router adjacent to the proxy. This router will remove the interface associated with the proxy from its forwarding state and stops sending any traffic from the proxy. As a result, routers downstream will timeout and remove the hidden path.

In Figure 4.3.1, suppose D sends an rprune for $P_1$. The message first reaches $R_1$, which is on the unicast path to $P_1$. $R_1$ verifies the message (using the hash anchor) and forwards it to $R_2$. Since $R_1$ is not directly connected to $P_1$, it does not take any action. When $R_2$ receives the rprune, it too verifies the message and finds out that it is directly attached to $P_1$ ($R_2$ can compare its directly attached subnets with that of the proxy's address). $R_2$ removes the input interface attached to $P_1$ from the set of input interfaces for $G$. Any packets sent by $P_1$ will be dropped at $R_2$. Due to this, invalid keepalives from $P_1$ will not reach downstream routers on the hidden path, and those routers will remove the state after timeout. In the example, $R_3$ is the downstream router (following the yellow) path towards the destination. Note that not only can this method be used to remove misbehaving proxies, but it can also be used to remove misbehaving routers. For instance, if $R_2$ does not comply with the destination's rprune request and the flood from the proxy still arrives at the destination. The destination can send an rprune for $R_2$. In this case $R_3$ will detect that it is directly attached to $R_2$ and remove the interface $1$ from the input interface set for $G$. As a consequence of removing $R_2$, the proxy $P_3$ will also be removed. However, this is acceptable since the misbehaving router $R_2$ would not have allowed $P_3$ to successfully communicate with the destination any way. Note that rprune can be use to remove the path to any misbehaving router as well.

### 4.3.7 Removing All Hidden Paths

Occasionally D may need to switch to a different $G$ and remove all the hidden paths associated with an old $G$. This may be because, 1) D ran into a collision with respect to $G$ after constructing hidden paths to a subset of the proxies, and 2) both a proxy and its adjacent router are compromised and D cannot remove the hidden path to the proxy. Although D may individually remove the hidden paths to the proxies, this process has message overhead and may take a while to complete. Moreover, the misbehaving proxy and its adjacent router will keep the state active at downstream routers. Therefore, to

remove all hidden paths D sends an explicit *purge* message to its upstream router. The format of this message is shown in Figure 4.3.2, and Algorithm 1 describes the router actions upon receiving a purge.

The purge message is addressed to the access router(s) D uses to reach the proxies. However, unlike the rjoin and rprune messages, which are addresses to a proxy, the purge message is addresses directly to $G$. In Figure 4.3.1, suppose D wants to remove all the hidden paths from the network. It sends a purge for $G$ to its upstream router $R_1$. $R_1$ consults its EFIB, verifies the message using the hash anchor, and sends the purge upstream on all interfaces found in the input interface set of $G$, i.e., $EFIB.G.\{if_{in}\}$. $R_1$ then removes the EFIB entry associated with $G$. When $R_3$ receives the PURGE, it too verifies the message and sends the purge upstream (on the yellow path) to $R_2$. Subsequently $R_3$ removes the state associated with $G$. Eventually the purge messages reach the proxies, and they too remove the state associated with $G$. Note that a misbehaving proxy or router(s) may not remove the state, but since downstream routers no longer have state associated with $G$, they drop any traffic sent to $G$.

**Algorithm 1** Functions at a router for processing various messages and creating the Epiphany forwarding table (EFIB)

**function** RJOIN($if_{join}$, $G$, $P_i$, $h_k$)
  $if_{pi} \leftarrow lookupUnicastFIB(P_i)$
  $if_{out} \leftarrow if_{join}$
  **if** $exists(EFIB(G))$ **then** // G is already claimed
    **if** $verify(h_k)$ **then** // Did D claim G?
      $EFIB(G).\{if_{in}\} \leftarrow (EFIB(G).\{if_{in}\} \cup if_{pi}) - if_{join}$
      $EFIB(G).if_{out} \leftarrow if_{join}$
  **else** // D claimed G first, create new entry for G
    $EFIB \leftarrow \langle G, \{if_{pi}\}, if_{out} \rangle$
  $setTimer(EFIB(G).\{if_{pi}\})$
  Update hash anchor $h_a$ with $h_k$
  send RJOIN Upstream on $if_{pi}$

**function** RPRUNE($G$, $P_i$, $h_k$)
  **if** $exists(EFIB(G)) \wedge verify(h_k)$ **then**
    $if_{pi} \leftarrow lookupUnicastFIB(P_i)$
    **if** $directlyAttached(P_i)$ **then** // Is proxy a neighbor?
      $EFIB(G).\{if_{in}\} \leftarrow EFIB(G).\{if_{in}\} - if_{pi}$
      **if** $EFIB(G).\{if_{in}\} = \phi$ **then**
        $EFIB(G) \leftarrow \phi$
    **else**
      send RPRUNE Upstream on $if_{pi}$
    Update hash anchor $h_a$ with $h_k$

**function** PURGE($if_{purge}$, $G$, $h_k$)
  **if** $exists(EFIB(G)) \wedge verify(h_k)$ **then**
    **if** $if_{purge} = EFIB(G).if_{out}$ **then**
      **for all** $if_i \leftarrow EFIB(G).\{if_{in}\}$ **do**
        send PURGE Upstream on $if_i$
      $EFIB(G) \leftarrow \phi$

**function** RECVDKEEPALIVE($G$, $if_i$)
  **if** $exists(EFIB(G)) \wedge if_i \in EFIB(G).\{if_{in}\}$ **then**
    $resetTimer(EFIB(G).\{if_i\})$

**function** TIMEREXPIRED($G$, $if_i$)
  $EFIB(G).\{if_{in}\} \leftarrow EFIB(G).\{if_{in}\} - if_i$
  **if** $EFIB(G).\{if_{in}\} = \phi$ **then**
    $EFIB(G) \leftarrow \phi$

**function** RECVDPACKET($G$, $if_i$)
  **if** $exists(EFIB(G)) \wedge if_i \in EFIB(G).\{if_{in}\}$ **then**
    $resetTimer(EFIB(G).\{if_i\})$
    Forward packet on $EFIB(G).if_{out}$
  **else**
    Drop packet

# Chapter 5

# Evaluation

In this section we evaluate the main mechanisms of Epiphany through simulations on an Internet scale topology, implementation on a local testbed and on a PlanetLab overlay. While we cannot offer a complete evaluation of the effectiveness of some aspects of Epiphany, such as reputation mechanisms, client access policies by the service and tiered DP implementation, we evaluate the new mechanisms such as the effectiveness of anycast in isolating clients from bots, the delay overhead involved in hidden paths and the effectiveness of separating DPs and SPs.

A key feature in Epiphany is the anycast SP mechanism. Anycast helps dilute the attackers and render them ineffective. In addition, the anycast SPs localizes the attacker, thus making the source network responsible for dealing with the attackers. Moreover, simpler techniques such as client puzzles [43] could be used to further limit the attacker. We simulate anycast SPs on an Internet scale topology using a simple model in which attackers, legitimate clients and SPs are placed randomly on the topology. Although the model may not represent the actual numbers of attackers or legitimate clients or their locations in the real Internet, it provides insight into how effective the anycast SPs will be in rendering the attackers ineffective. The simulation also evaluates different strategies for placing the SPs and how those strategies affect the attackers and legitimate clients.

Another important feature in Epiphany is the ability to quickly add and remove proxies to handle different scales of attacks. However, this requires quickly constructing and removing hidden paths. We demonstrate this design feature using a simple implementation of Epiphany on Cisco routers using single source multicast (SSM) [14]. Although

this implementation is not the exact same mechanism by which Epiphany creates hidden paths, it is however we envision Epiphany hidden paths to be implemented on routers. Using this testbed we measure path setup and teardown delay and provide estimates on what to expect when constructing/tearing down hidden paths in Epiphany. These measurements are essential to understand the expected delays when adding and removing proxies.

Finally, Epiphany separates the SPs from DPs to minimize disruptions to established communications. Even though the attacker is targeting an SP, once a client successfully sends a request through and establishes communication with the destination, it will not be affected by any amount of attacks on the SPs. This approach is different from using the same proxy for both setup and to communicate with the service. We implemented Epiphany as an overlay running on top of PlanetLab and use this overlay to measure the effectiveness of separating the proxies, and also to evaluate setup delays.

## 5.1 Effectiveness of Anycast SPs

We conduct simulations to evaluate the effectiveness of anycast SPs in isolating attackers from legitimate clients on the 2011 dual Router+AS topology from CAIDA [10]. To this end, we developed a router+AS topology simulator to study anycast SPs under: 1) various attacker and legitimate client distributions, and 2) different scenarios for placing SPs. The simulator does not consider link capacities, because such information is unavailable with the topology.

### 5.1.1 Methodology

We first preprocessed the CAIDA ITDK [10] topology data (which is contained in nodes and links files) to obtain an undirected router graph $G(V, E)$, where $V$ is a set of routers and $E$ is the set of undirected edges. To this end, we chose the most accurate alias re-

solved router topology, namely midar-iff, and combined the information from the nodes file and links file to derive a list of routers and their respective interfaces. The topology data does not include router interfaces that connect to end hosts. Therefore we assume any router with only one interface to be connected to end-host subnets. We converted the IP level connectivity information provided in the links file to graph edges. For example, if router 1 shares an IP link with routers 2, 3 and 4; we create the following graph edges: {1 - 2}, {1 - 3}, {1 - 4}, {2 - 3}, {2 - 4}, {3 - 4}. Next, we extract the largest connected component from the topology to form $G(V, E)$. The resulting graph has $\approx 3.3$ million routers and $\approx 21$ million edges.

To derive the AS level graph, we took the router to AS mapping provided in the topology data and converted the router level graph $G(V, E)$ to an AS level graph $G'(V', E')$, where $V'$ is a set AS nodes and $E'$ is the set of AS level undirected edges. We extracted the largest connected component from this graph. The resulting AS graph has 20,115 nodes and contains 2,743,974 routers (4 AS nodes were eliminated in this process). Some routers that are part of the router level graph were not part of the AS level graph because they either did not have an AS mapping, or they were not part of the 20,115 AS nodes. We pick our attackers, legitimate clients and SPs from these 2,743,974 routers, but use the full 3.3 million router graph to compute path information. Of the 2,743,974 routers in the AS graph, 1,356,359 routers only have one interface and the remaining 1,387,615 routers have more than one interface. We denote the routers with only one interface as *edge routers*, and the rest as *core routers*. Note that some routers may have multiple interfaces that connect to end-host subnets (for instance, multi-homed routers). However, since we do not have the subnet information, we treat these as core routers in the simulation. We always choose the attackers and legitimate clients from the edge routers, but place the SPs on either or both edge and core routers.

We model our attackers and legitimate clients in the topology using a simple ap-

proach; we randomly pick $N_a\%$ of the edge routers and denote them as attackers, and similarly pick $N_l\%$ of edge routers and denote them as legitimate clients. Our model is very general in that it does not consider specific distribution of actual attackers versus legitimate clients, or their locations in the Internet. However, in the simulations we vary the percentage of attackers, legitimate clients and SPs to study properties such as how many attackers are rendered ineffective, how many attackers interfere with legitimate clients, how far away are the attackers interfering with legitimate clients located from an SP, and so forth. Finally, we randomly pick $N_{sp}\%$ of all routers and denote them as SPs and assign them to attackers and legitimate clients based on the following rules:

- If there are SPs in the same AS, assign the nearest (in terms of router hops) SP to each of the attackers and legitimate clients.
- If no SPs exist in the local AS, pick the nearest remote AS that has SPs and assign the nearest SP in the remote AS to each of the attackers and legitimate clients.

### 5.1.2 Varying Attacker Distribution

We first vary the attacker distribution from $N_a = 0.1\%$ to $N_a = 10\%$ with a fixed legitimate client percentage $N_l = 0.01\%$. In other words, the simulation models attackers that are 10, 100 and 1000 times stronger than the legitimate clients. We show the results for various SP densities. For each data point, we run the simulation 10 times and plot the average with error bars.

Figure 5.1.1 shows the percentage of attackers that are on SPs not used by legitimate clients i.e., these attackers cannot aggregate their traffic to target specific SPs. If the destination is able to detect them and tell the corresponding SPs to disable accepting requests (as discussed in Section 4.2.3), these attackers would become ineffective and can no longer harm legitimate clients. The graphs show that even at 0.025% SP density, almost 50% of the attackers are neutralized *regardless of the attacker distribution*. How-

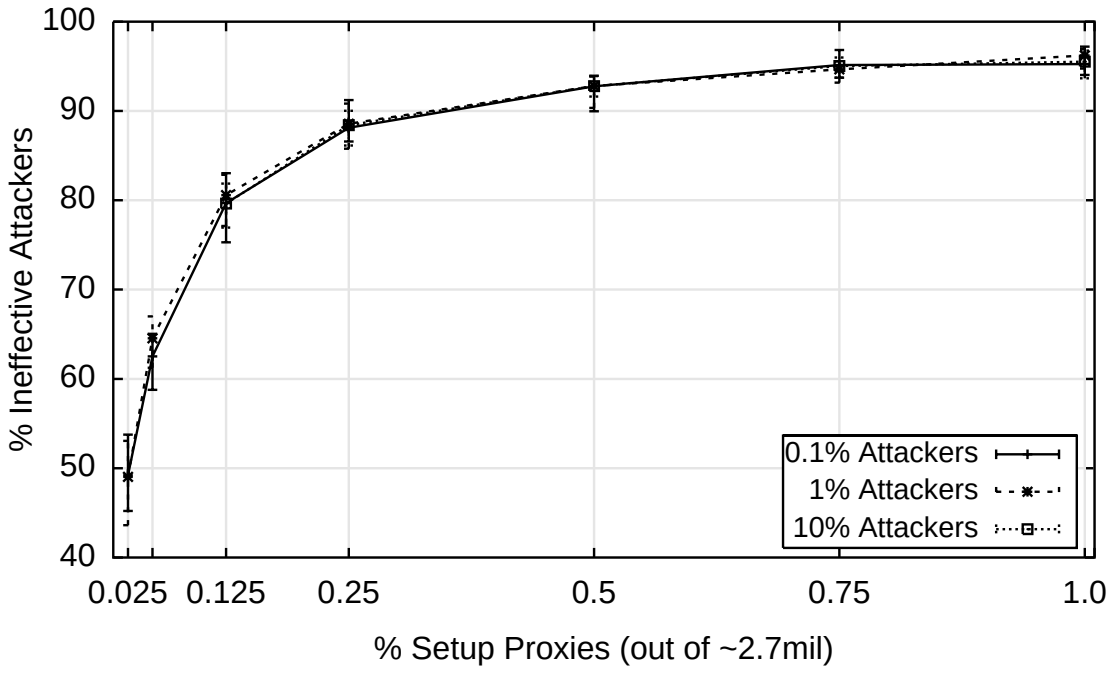


**Figure 5.1.1:** Percentage attackers rendered ineffective [$N_a = 0.1, 1, 10\%$; $N_l = 0.01\%$]

ever, at 0.5% SP density, about 93% of the attackers are neutralized. Increasing the SP density to 1% only marginally improves the number of attackers rendered ineffective. On the other hand, below 0.25% SP density, the attackers rendered ineffective starts to quickly drop off with smaller SP densities. In other words, at 0.5% SP density only about 7% of the attackers are interfering with the legitimate clients.

In contrast, Figure 5.1.2 shows the percentage of legitimate clients that do not have attackers on their SPs, i.e., these are the clients that will not be affected by any attackers. At 0.5% SP density and 0.1% attackers, nearly 63% of the legitimate clients become free from attackers. However, at 0.5% SP density and 1% attacker distribution, only 13% of the legitimate clients are unaffected by the attackers. Increasing the SP density improves the percentage of unaffected legitimate clients, but only slowly. This effect is more discerning at 10% attacker distribution. In this case, the percentage of unaffected legitimate clients starts at a meager 1% at 0.25% SP density and only increases marginally to 3%

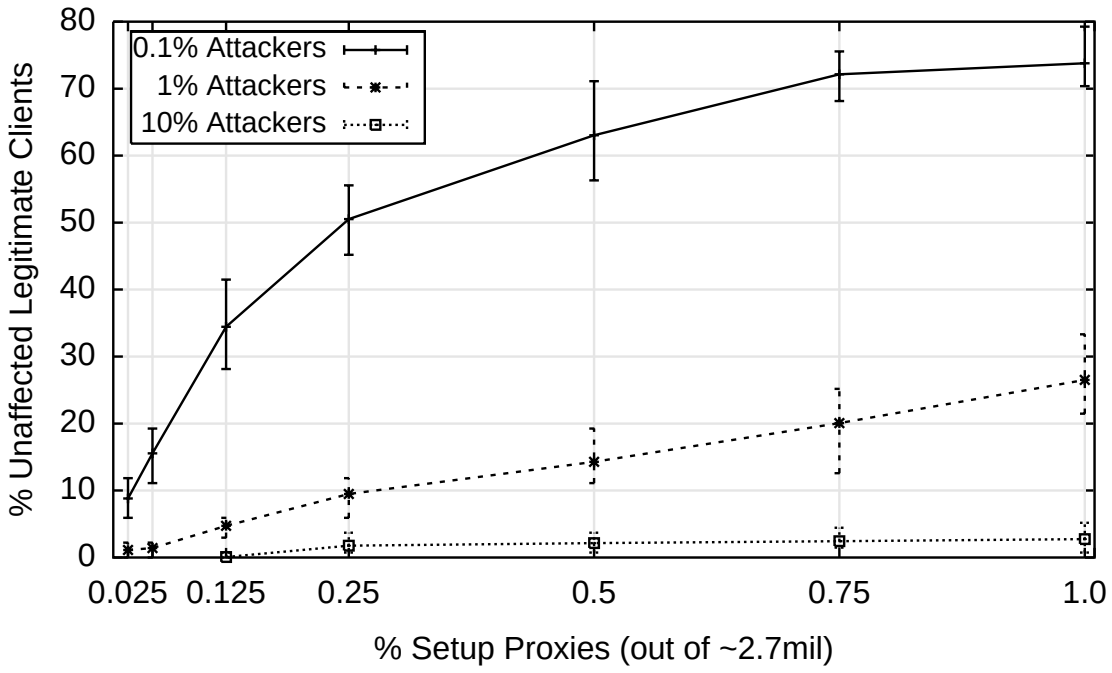


**Figure 5.1.2:** Percentage legitimate clients unaffected by attackers [$N_a = 0.1, 1, 10\%$; $N_l = 0.01\%$]

as more SPs are added. These numbers may seem not promising, since only a small percentage of legitimate clients are unaffected by attackers. However note that we are modeling a highly distributed attacker (in both $N_a = 1\%$ and $N_a = 10\%$). Moreover, our SPs are randomly placed rather than carefully planned and installed. In our model, as the attacker distribution increases, the probability that some attackers land very near in terms of number of hops to the legitimate clients increases. Now to separate them, two SPs must be placed in such a way that the attacker and legitimate client are routed to a distinct SP. When the distance between the attacker and legitimate client is greater, there are more routers in the topology between the two nodes to randomly place SPs so that they are separated, but as the distance becomes less, there are fewer choices and hence requires a much higher SP density to achieve the separation. We observe this with the $0.1\%$ attacker distribution. At $0.25\%$ SP density, about $50\%$ of the legitimate clients are unaffected by the attackers. That is because, at that density there are 5 times more

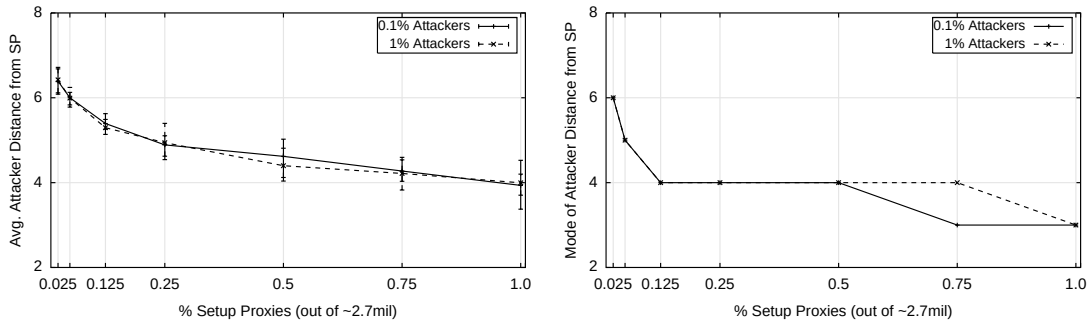


**Figure 5.1.3:** Average and mode of the distance from attackers affecting legitimate clients to the SPs [$N_a = 0.1, 1\%$; $N_l = 0.01\%$]

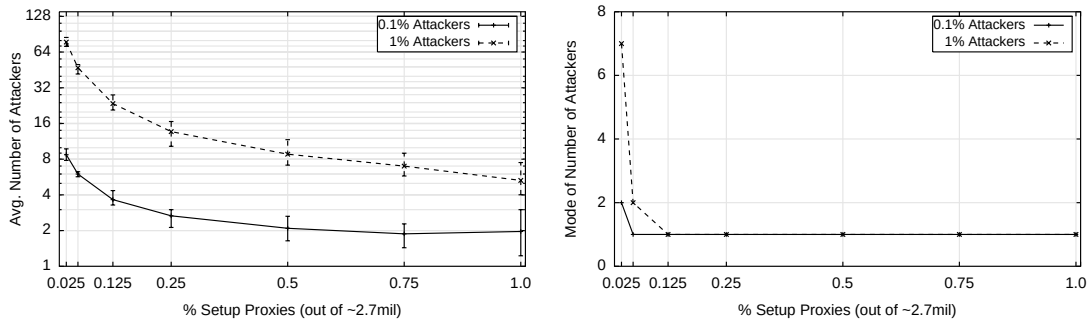


**Figure 5.1.4:** Average and mode of number of attackers affecting legitimate clients [$N_a = 0.1, 1\%$; $N_l = 0.01\%$]

SPs than attackers and legitimate clients combined ($\frac{2743974 \times 0.0025}{1356359 \times 0.00101} \approxeq 5$). Adding more SPs certainly increases the percentage of unaffected legitimate clients, but only up to a certain point. In particular, the number of unaffected legitimate clients (at $0.1\%$ attacker density) marginally improves from $72\%$ to $73\%$ going from $0.75\%$ SP density to $1\%$ (at $1\%$ SP density there are 20 times more SPs than attackers and legitimate clients combined). We believe at this point, most of the remaining legitimate clients are very close to the attackers and hence separating them requires a much higher SP density.

Figure 5.1.3 shows the average and mode of the distance from the attackers that interfere with legitimate clients to their respective SPs. As the SP density increases, the average distance between the attackers and the SPs decreases. However, the mode shows that most of the time the attackers are only 3 to 4 hops away from the SPs after

about 0.125% SP density. On the other hand, Figure 5.1.4 shows the number of attackers that interfere with the legitimate clients at an SP. With fewer SPs, the average number of attackers interfering with legitimate clients is very high, but as more SPs are added, this average drops quickly. For instance, at 1% attackers and 0.5% SPs, on average there are only 8 attackers interfering with legitimate clients, but the mode shows that in most SP regions, there is only one attacker interfering with legitimate clients.

In summary, regardless of the attacker distribution, with a small legitimate client density of 0.01% and an SP density of 0.5%, over 93% of attackers are are neutralized. Moreover, at that SP density, *most often there is only one attacker affecting a legitimate client*, and this attacker is only 3 to 4 hops away from the SP. In other words, the SPs effectively localize the attackers to a single subnet which is often very close to the SP. These results suggest that the anycast SPs make the techniques to locate attackers simpler, since there are far fewer attackers to find and they could be found in a smaller network radius. Moreover the additional techniques to further limit the attackers discussed in Section 4.2.4 could be employed effectively.

### 5.1.3 Varying Legitimate Client Distribution

We now hold the attackers at 1% and vary the legitimate client distribution from 0.01% to 1%. That is, the legitimate clients are 10 to 100 times smaller than the attackers, and equal to the number of attackers. Again, we conducted the simulation 10 times for each data point and show the average with error bars. The results are shown for increasing SP densities.

Figure 5.1.5 shows the percentage of attackers rendered ineffective at different legitimate client distributions. As the legitimate client density increases, the percentage of attackers rendered ineffective for a given SP density also decreases. Because now, with the increased number of legitimate clients, the likelihood that clients end up in attacker regions increases, thus fewer attackers become ineffective. For instance, when

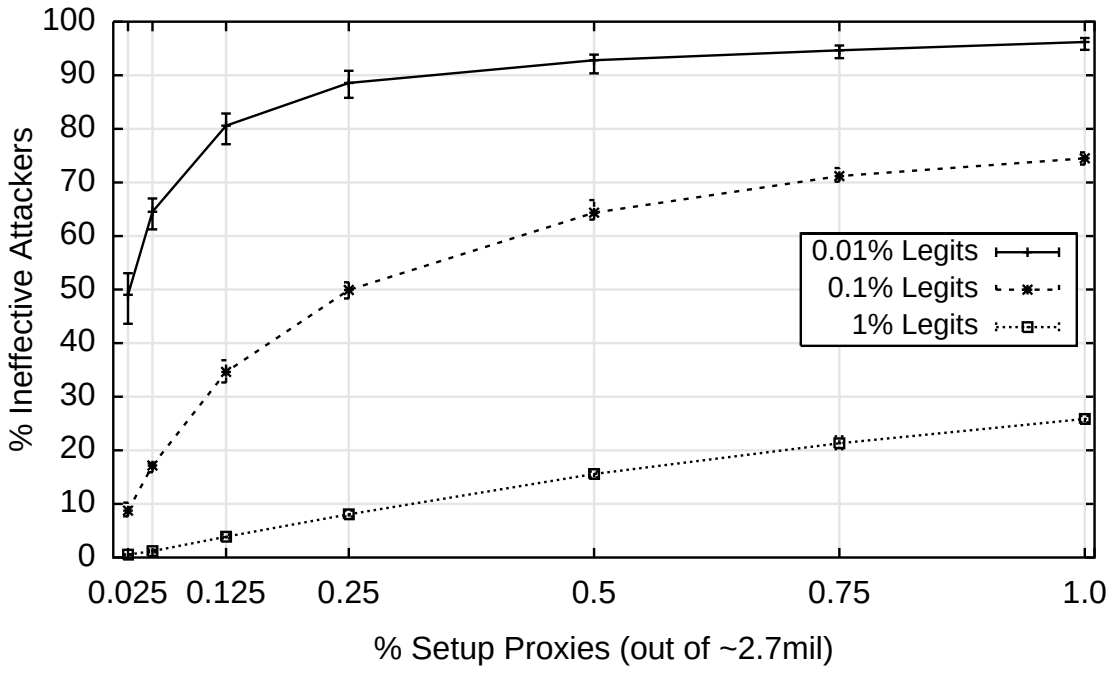


**Figure 5.1.5:** Percentage attackers rendered ineffective [$N_l = 0.01, 0.1, 1\%$; $N_a = 1\%$]

the legitimate clients and attackers are both 1%, the probability that they end up closely located to each other increases and may end up being routed to the same SP. As explained earlier, separating the attackers from legitimate clients when they are closely located requires higher SP densities. Figure 5.1.6 shows the percentage of legitimate clients that are unaffected by attackers. Here, the percentage of legitimate clients unaffected remains mostly similar across different legitimate client distributions for a given SP, attacker distribution. For example, at 0.5% SP density approximately 15% of the legitimate clients are separated from attackers. As the SP density is increased, more legitimate clients become separated from attackers. This is because, at a given attacker distribution, as more clients are randomly chosen, some of them may land very close to attackers (a real world analogy is that they come from unclean networks) while some land in SP regions without attackers (analogous to clean networks).

In summary when the attackers and legitimate clients are equally and heavy dis-

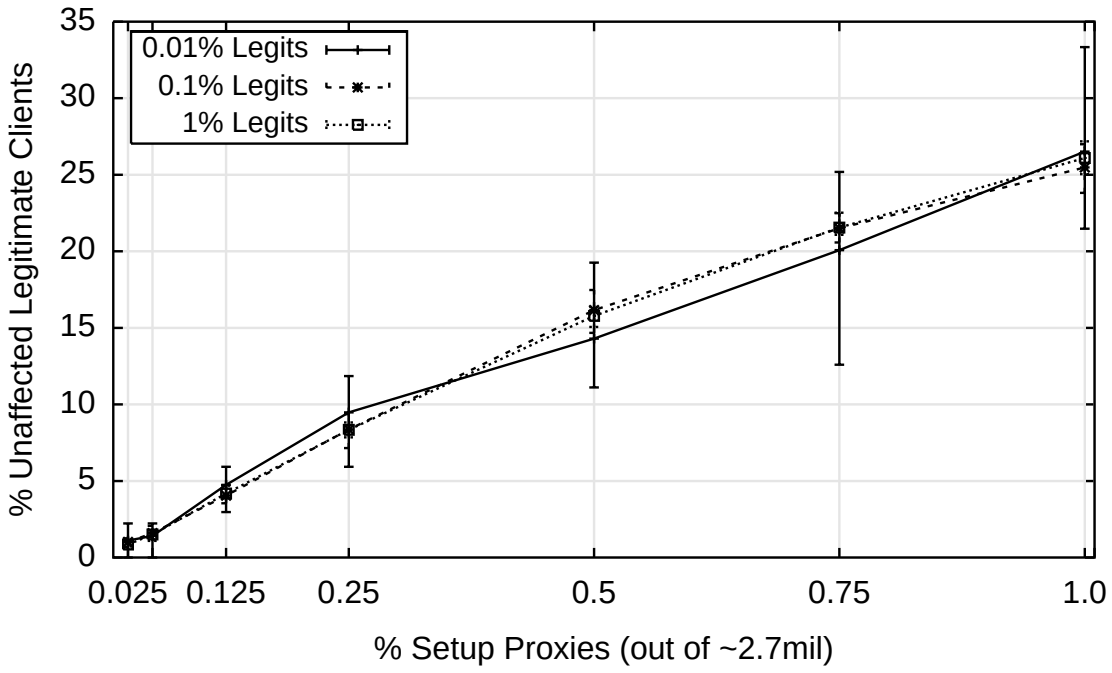


**Figure 5.1.6:** Percentage legitimate clients unaffected by attackers [$N_l = 0.01, 0.1, 1\%$; $N_a = 1\%$]

tributed, separating them requires a much higher SP density. For example, when $N_a = 1\%$ and $N_l = 1\$$, even at $1\%$ SP density only $25\%$ of the attackers are rendered ineffective and similarly only $25\%$ of the legitimate clients are unaffected by attackers.

### 5.1.4 Effects of Various SP Placement Scenarios

Through this next set of simulations, we wish gain some insight into different strategies for placing SPs. The scenarios are based on choices a critical service provider might face and/or depending on the availability of SPs. *1-SP-Per-AS* examines the case when a critical service provider wants to establish regions based on AS boundaries, i.e., *1-SP-Per-AS* attempts to place at least one SP in each AS, but if there are fewer SPs (than the AS nodes) they are placed at random AS nodes. SPs at the *Core* examines the case when SPs are operated by Internet Service Providers (ISPs) or site admins, i.e., in this method, SPs are placed at random core routers. *Edge* examines deploying SPs at the edges, on

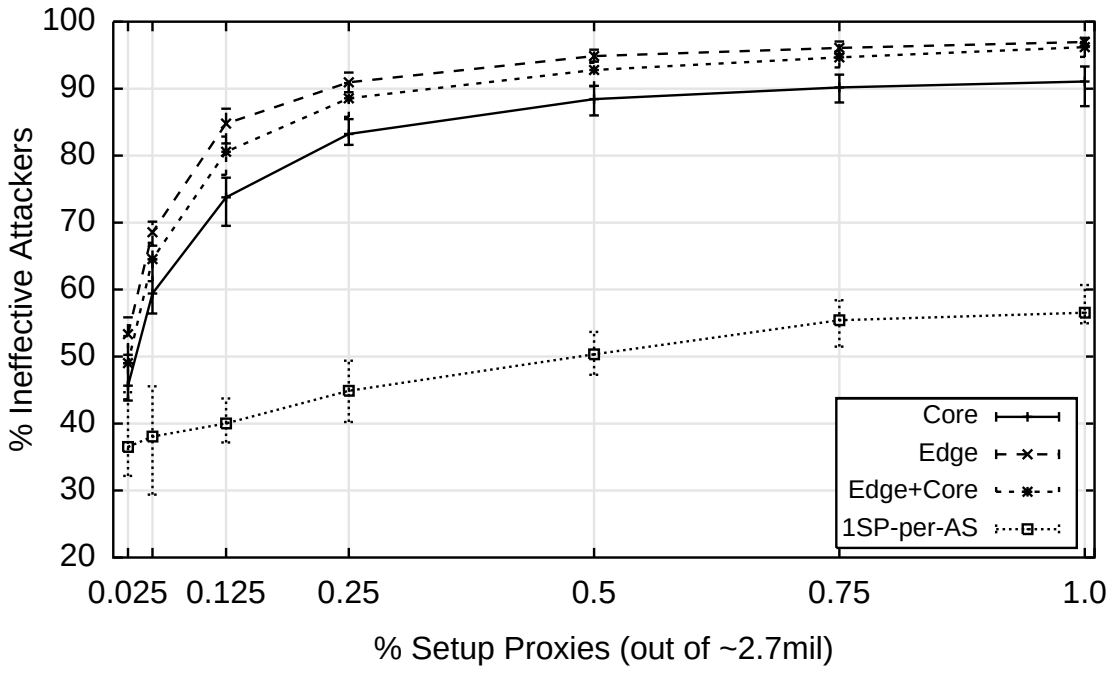


**Figure 5.1.7:** Percentage attackers rendered ineffective with $N_l = 0.01\%$, $N_a = 1\%$, and with various SP placement scenarios

perhaps end user machines. Finally, *Edge+Core* examines a general case in which the service provider has access to some SPs at the edge and some at the core across different AS nodes.

Figure 5.1.7 shows the percentage of attacker rendered ineffective, while Figure 5.1.8 shows the percentage of unaffected legitimate clients when $N_l = 0.01\%$, and $N_a = 1\%$. From the various SP placement strategies, placing the SPs on edge routers provides better separation of attackers and legitimate clients. For instance, at 0.5% SP density, the edge only approach rendered 93% attackers ineffective, while Core, Edge+Core and 1SP-Per-AS rendered 95%, 89% and 50% attackers ineffective. As we mentioned earlier, separating the attackers from legitimate clients requires placing SPs in a way that they use distinct SPs. In Edge, this is more likely because an SP may land on the same edge router as an attacker or a client. In contrast, in the *Core* only placement scenario, an SP is not placed on the same edge router as an attacker or a legitimate client and

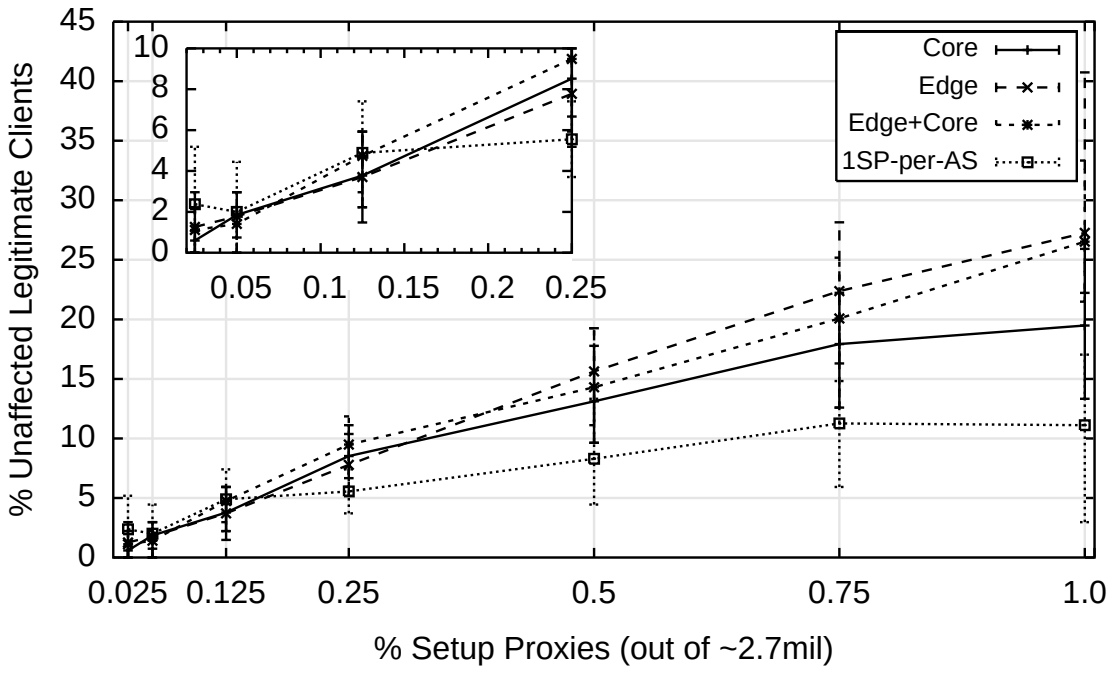


**Figure 5.1.8:** Percentage legitimate clients unaffected with $N_l = 0.01\%$, $N_a = 1\%$, and with various SP placement scenarios

therefore it becomes harder to separate them. However, in the Internet, placing all SPs on the edge or core only might not be possible depending on the availability of SPs (some edge networks might not provide any SPs). The *Edge+Core* scenario is more likely and presents a balance between the edge only and core only scenarios.

Finally, placing the SPs one per AS does worst, i.e., at 0.5% SP density, only 50% of the attackers are rendered ineffective. In addition, increasing the SP density has marginal effect in terms of separating attackers from legitimate clients. This is because some SPs are wasted in ASes that might not have either attackers or legitimate clients, while in the ASes that do have them, the attackers and legitimate clients are tied to the same SP. Interestingly, when there are far fewer SPs in the network, the *1-SP-Per-AS* scenario is able to separate more legitimate clients from attackers compared to the other scenarios, i.e., for instance at 0.025% SP density, 5% legitimate clients are unaffected by attackers, while Edge only and Core only have 3% legitimate client unaffected by attackers (see

Figure 5.1.8). We believe this is because the SPs in 1SP-Per-AS scenario are able to roughly divide the network in such a way that a few regions have only attackers, and a few other regions have only legitimate clients (but a large portion of the regions have both attackers and legitimate clients).

In summary, placing the SPs on *Edge+Core* offers the best compromise in terms of separating the attackers from legitimate clients. However, when there are far fewer SPs available (for instance when a service is just starting to roll out) or when the critical service provider is only able to acquire SPs in distinct ASes, then *1-SP-Per-AS* seems a better compromise than the other scenarios at smaller SP densities.

## 5.2 Implementation on a Router Testbed

Epiphany hidden paths are a new concept, but the basic functions to implement some of the functionality of hidden paths are already available on routers in the form of multicast routing. Specifically, we implement a variant of hidden paths on cisco 2600 routers using single source multicast (SSM) [14]. Figure 5.2.1 shows our experimental testbed. The SP, DP, source and destination are off-the-shelf commodity hardware based on Intel Pentium 4 processors (with 2Gb RAM) and are running Linux OS. All the forwarding functions at the SP and DP are implemented as user level programs, while the source and destination are Epiphany aware applications which simply transfer fixed size UDP messages. We setup static routing so that the source and the proxies can reach each other, but the destination prefix in not announced to any other router. In other words, there is no way for either the proxies or the source to reach the destination even if they know the destination's IP address.

We use SSM for two reasons. First, in SSM, like any other multicast protocol the sender does not know the receiver's address, and second, the receiver can leave the multicast group at any time to stop receiving further traffic for that group. SSM also

allows the destination to explicitly specify which sender it wishes to receive the traffic from. Without this control, any host sending to a multicast group that the destination specifies interest in will be able to send traffic to the destination. We exploit these features of SSM to implement hidden paths. Initially, D randomly picks a multicast group G1 and specifies interest in this group using the IGMPv3 membership reporting protocol [11]. Since the destination only wants to receive packets from the SP, D sets up an IGMPv3 filter and includes the SP as the sender for this group. The IGMPv3 protocol reports this interest to the adjoining router R1, which is on the path to the SP. The IGMP message has the destination's IP address, but R1 only uses it locally and never reveals the destination address to any other router (in fact, R1 never announces a route for D's address prefix to any other router). In response to the IGMP membership interest, R1 creates a multicast forwarding entry <SP, G1>, and sends an SSM *join* message upstream to R2 after identifying the interface on which it should send the join using the SP's unicast address. The router R2 similarly creates <SP, G1> forwarding entry, but does not propagate the message any further since the SP is directly attached to it. Note that the multicast forwarding entry created by the join message is different from Epiphany's EFIB entry, but a few similarities exist between the two. Epiphany does not care which source is sending the packets, as long as they arrive on one of the input interface set; if they arrive on an interface not belonging to this set, they are dropped/filtered immediately. While in SSM, routers maintain explicit state about which sources are allowed to send traffic to a particular multicast group. Packets arriving from a source that is not in the list of allowed sources (for a particular multicast group) are dropped immediately. To create hidden paths to all proxies using SSM, a router would require memory resources proportional to the number of proxies, while in Epiphany there is only one entry per destination.

In order for the SP to forward requests to the destination it needs to know about G1.

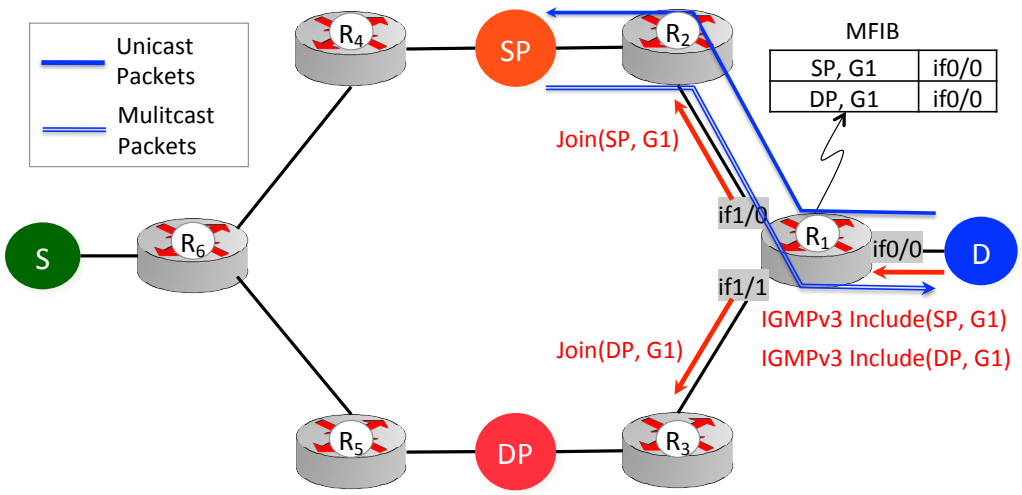


**Figure 5.2.1:** Experimental testbed implementation of Epiphany on cisco 2600 routers and commodity hardware

After sending the IGMPv3 membership message, D sends a unicast message to the SP with G1, its own identifier (i.e., D's ID), and the rate limit for requests. The SP creates a lookup table entry with this information. Note that the destination always sends packets to the proxies using their unicast IP addresses. When a source S sends a request to the SP, the SP looks up D (i.e., D's identifier), checks the rate limit (using a token bucket algorithm), and forwards the request to the destination by wrapping it in a multicast packet addressed to group G1. When the request arrives at router R2, the router lookups G1 in the multicast forwarding information base (MFIB) table and forwards it to R1. The router R1 similarly looks up G1 and forwards the message to D. The destination accepts the request and sends an SSM join for {DP, G1}. The destination then sends a unicast message to the DP informing about G1, the rate limit for S, and the authorization token to verify S. The source learns about the DP and sends its data through the DP. When the destination wants to remove the hidden path to a proxy, say the SP, it sends an IGMPv3 membership report message to exclude the SP from G1. D also stops reporting membership interest for G1. After a timeout interval, router R1 sees that it no longer has members interested in G1 and sends a *prune* message for G1. At the same time, R1

also removes the multicast forwarding entry for G1. The prune message propagates to the router R2, which then stops forwarding multicast packets for G1 (and removes its own MFIB entry).

### 5.2.1 Measuring Creation and Removal Delays

Using this simple testbed, we measure the delay to setup and teardown the path between the destination and the SP. These measurements give an estimate of the delays to expect when constructing and tearing down hidden paths in Epiphany. First, to measure the path setup delay, we make the destination send a unicast message with the address G1 that the SP could use to reach D. Immediately following this message, D initiates the hidden path construction process (as outlined above). Once the SP receives the message from D, it immediately starts sending Epiphany request packets. The destination measures the delay between the time it sent the unicast packet to the time when it receives the first request packet from the SP. This delay includes the time to setup the path and the round trip time taken for the unicast message from D to reach the SP, and the request from the SP to reach D. Note that some requests from the SP will be lost as the path is setup. We measure the round trip separately (using a series of messages between SP and D) and subtract it from the total time computed above to get the path setup time. We repeated the experiment 20 times and averaged the results.

From the experiments, we observed that the path setup took about $13.89$ milliseconds, while the round trip time is only $1.18$ milliseconds. Here, the round trip time is composed of the delays to transmit, propagate, and process the packets at routers. In our simple testbed, these delays are minimal due to the short length of physical cable involved, and the low load on routers. In the Internet however, the propagation delays will dominate the round trip time and processing delays at router may further increase this delay, therefore the setup time in the Internet will be dependent on the total propagation time of the hidden path create message. However, based on current estimates of

round-trip times, we believe these delays will be well within the milliseconds range.

Next, to measure the teardown delay, we make the destination send a message to the SP to tell it to start sending requests at a constant rate of 200 requests per second. That is, the SP sends requests at every 5 millisecond interval (we chose this interval due to the relatively low timing accuracy on our hosts). Immediately following this, D sends an IGMPv3 membership report message to exclude the SP from group G1, i.e., D expresses interest to remove the SP from sending any packets to D. The router responds to the membership report message by stopping to forward any packets from the SP. However, the router does not generate a *prune* message until it finds that it no longer has any members interested in G1. This duration is a random interval in the order of seconds. During this time packets from the SP still arrive at the destination network. We use `tcpdump` tool to capture the traffic and analyze the number of packets that arrived at the destination after the path removal was initiated. We then compute the delay using: number of packet received after teardown $\times$ send interval. For example, if 200 packets arrive at the destination after the path removal is initiated, then the teardown delay is $200 \times 5 = 1000$ milliseconds, or 1 second. We repeated the experiment 20 times and averaged the results.

We observed that the path tear down took about $1276$ milliseconds on average. Since the round trip time is minimal it does not affect the teardown delay. However, our accuracy is under 5 milliseconds due to the SP's sending rate, i.e., the path would have been removed anytime between $1276$ milliseconds to $1281$ milliseconds. In the Internet, the propagation delays will increase this delay, but they are generally smaller compared to this delay. In other words, removing hidden paths might take in the order of seconds. During this time, a compromised proxy may cause some damage to legitimate traffic by flooding the destination, but the damage will be limited to within few seconds.

Another important lesson we learned from the testbed is that the destination network

prefix need not be exported to the global Internet. This an important finding, since no other location hiding architecture has suggested completely hiding the destination address and also making it impossible to reach the destination without having an explicit hidden path.

In summary, we have demonstrated how to create and hidden paths using existing router functionality. While more functionality would be nice to have (such as an explicit path teardown process), we have demonstrated the viability of hidden paths within existing router functionality. This gives us confidence that the Epiphany mechanisms are both feasible and deployable.

## 5.3 Implementation on PlanetLab Testbed

We implemented the main Epiphany components as an overlay on top of PlanetLab [3] to evaluate setup and data transfer time in a real Internet setting. These experiments also provide evidence of the effectiveness of separating setup and data proxies. The implementation consists of a central *topology server*, which is responsible for creating and managing the overlay, and an overlay *node application* which implements the functions of an Epiphany router (R), source (S), destination (D), setup proxy (SP), and data proxy (DP). Our implementation consists of approximately 6000 lines of C++ code to implement the Epiphany components, and another 2000 (approximately) lines of Perl code to implement the topology server.

### 5.3.1 Methodology

The topology server takes as input the number of Epiphany routers, SPs, DPs, and a list of PlanetLab hosts. It then randomly selects some hosts from the list and launches the node application instances for the routers, SPs, DPs and D. The topology server also gives the router hosts a list of all other routers in the overlay. The router hosts

use this list to ping each other and inform the topology server about the Round Trip Time (RTT) to the other routers. The topology server gathers this information and for each router computes at least three other neighbors that have the shortest RTT to form a connected overlay graph. If the resulting graph does not form a single connected component, the topology server selects additional neighbors for some router to form a connected graph. After this step, the topology server gives the connectivity information to the routers (i.e., list of router destinations and the neighbors to reach them), which then form the overlay using TCP connections. Next, the node applications for SPs, DPs and D ping the routers and attach to the closest router in the overlay using RTT as the metric. The destination also learns about the SPs and DPs from the topology server. After the overlay has initialized, the destination sends rjoin messages to the proxies to construct hidden paths. The destination also sets the rate limit $R_R$ for the number of requests each SP can forward to D in a second. Once this process is complete, the topology server launches a specified number of attackers and legitimate clients. These sources compute the closest SP using the RTT value and start sending requests at rates $r_l$ and $r_a$ respectively, for a duration of $T_{run}$ seconds. We ensure that the attackers always start sending requests to the SPs before the legitimate client and end after the legitimate client stops sending requests (or data). We do not implement real attacks, where a PlanetLab host (attacker) is flooding another PlanetLab host (SP), but rather emulate them by making the attacker hosts send at sufficiently high rate. To implement rate-limiting at the SPs and DPs, we borrowed the token bucket implementation from click [2]. Also, since our node applications use TCP, we set the TCP_NODELAY option to avoid potential buffering (due to nagle's algorithm) at the sender, which could skew results. We also followed the best practices described in [37] for running the experiments on PlanetLab. In particular, [37] notes that generating packet streams on PlanetLab nodes at precise intervals is difficult, especially for high rates. Therefore our rate-limits

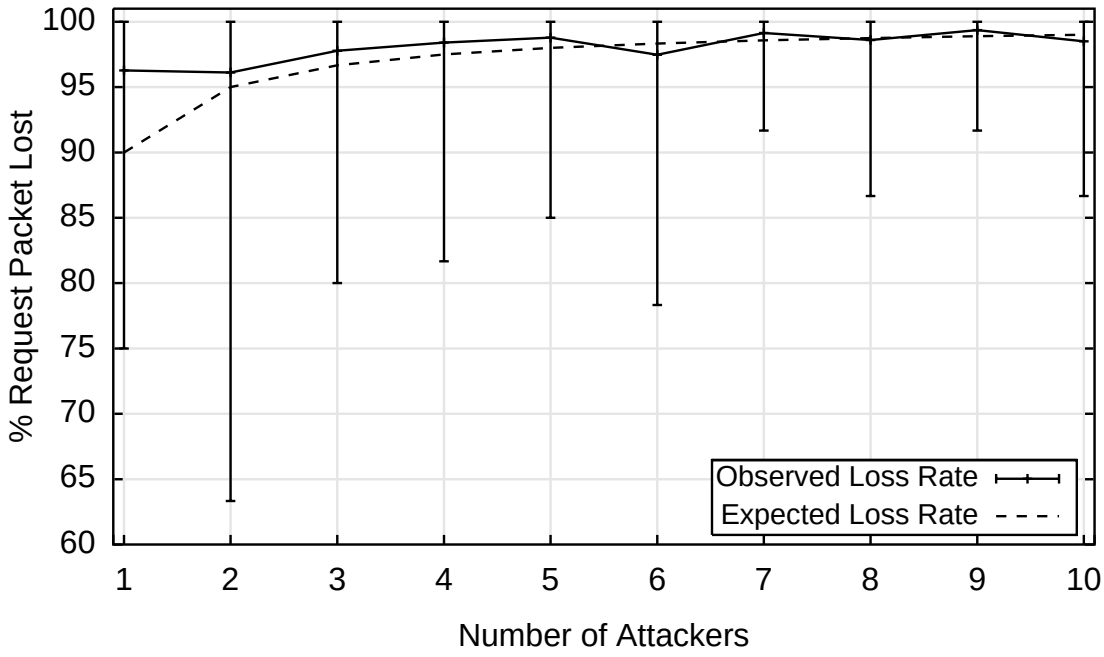


**Figure 5.3.1:** Percentage requests lost of a legitimate client with 1 to 10 attackers and 1 SP

are set in such a way that we do not encounter these pitfalls.

### 5.3.2 Request Losses at SPs

We first study the request losses experienced by a legitimate client when both the attackers and legitimate client are on the same SP. We assume the SP does not employ any additional defenses such as the ones described in Section 4.2.4. Figure 5.3.1 shows the request loss percentage for 1 legitimate client with an increasing number of attacker hosts (form 1 to 10). Each attacker sends at rate $r_a = 100$ Requests Per Second (rps), while the legitimate client sends at $r_l = 1$ rps. The request rate-limit $R_R$ at the SP is set to $10$ rps. In the Internet, the destination may not know how many clients are behind a specific SP and therefore it has to accommodate several legitimate clients sending requests at once. Our estimate of $R_R$ in this case approximates a small subnet (i.e., it can accommodate 10 legitimate clients sending requests at once). We run the experiment

for a total duration of 3600 seconds, but take periodic samples at every 60 second intervals. Figure 5.3.1 shows the expected and average percentage of requests lost in an interval due to the attackers, while the error bars show minimum and maximum percentage of requests lost in a sample interval. The expected loss percentage is computed using: $1 - (\frac{r_l}{r_a \times n_a + r_l} \times R_R)$. From the figure, it is clear that although the expected loss percentage should be $90\%$ with only one attacker, the dynamic behavior is much worse. Numerous factors contribute to the dynamic behavior, including delays in the network which affect the arrival time of the legitimate request, coalescing of legitimate requests in the network and so forth. However, in any case since the attacker is sending at a much higher rate the legitimate requests will be dropped at the SP due to the rate-limit. As the number of attackers increase, the losses also increase as expected. In other words, without the additional defenses we described in the design, a legitimate client cannot expect to successfully reach the destination in an expected amount of time when the SP it is using is also occupied by attackers.

### 5.3.3 Setup & Data Transfer Delays

Now that we have established the request losses of a legitimate client in the presence of attackers, we ask ourselves "at what attack rates is a legitimate client likely to reach the destination within a reasonable amount of time?" This is an important factor to study, since once the defenses we discussed in Section 4.2.4 have been established, attackers may not be able to completely block the legitimate clients from reaching the destination but may seek to delay them for prolonged periods. Studies on Internet users found that a significant percentage of users will wait no longer than 3 seconds to receive the service after initiating a request [1]. However, since Epiphany is protecting a critical service, we assume the users in Epiphany are willing to wait longer, i.e., in the order of a minute to receive the service. The first part of our experiments study the setup delays at different attacker rates. Then, we study the data transfer delays experienced in

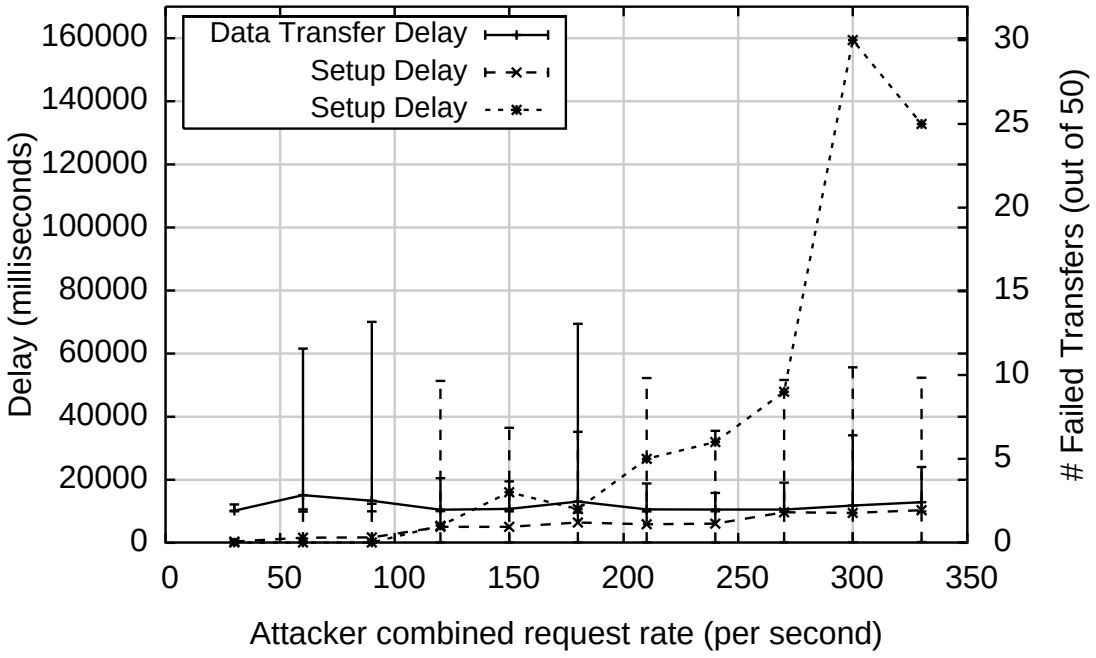


**Figure 5.3.2:** Setup and data transfer times of a legitimate client with 1 SP and 1 DP and an increasing attacker rate. A total of 15 attackers are used, while the setup rate limit is fixed at 50 rps.

Epiphany and compare it against the delays experienced when a single proxy performs both data transfer and setup functions (such would be the case in SOS [17]). Since SPs and DPs are different nodes in Epiphany, we shown the delays separately in the graphs.

In the following experiments, we use 15 different attacking hosts to generate a target attack rate. For example, to achieve an attack rate of 150 rps, each attacking host sends 10 requests per second. This is inline with our notion that if the SP is employing cryptographic puzzles, the attacker may strategize and use multiple hosts to generate requests with valid puzzle solutions at a constant rate. In all the experiments, a single legitimate client attempts to send 1000 data packets of 100 Bytes each to the destination. The request rate-limit $R_R$ is set to 50 rps at the SP, while the data rate-limit $R_D$ is set to 100 Data Packets Per Second (dps) at the DP. For each data point, we repeat the experiment 50 times and compute the average delay (with minimum and maximum

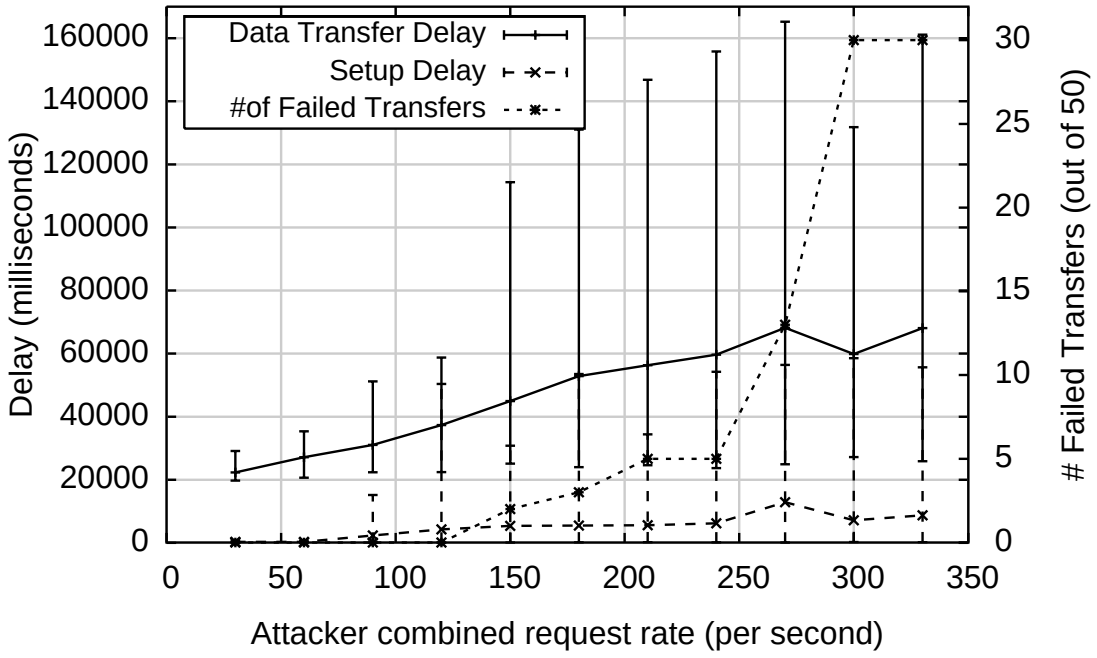


**Figure 5.3.3:** Setup and data transfer times of a legitimate client when a single proxy acts as both SP and DP. The combined rate limit at this proxy is fixed at 50 rps.

delays shown using error bars). To measure the setup delay, we compute the time from when the legitimate client initiates a transaction to the time it receives a response from the destination. The client retransmits requests every one second for up to 60 seconds before giving up on a transaction (i.e., if it does not receive a response). We consider the transactions that do not receive a response within 60 seconds as failures. Note that the retransmission rate of the legitimate client in our experiments is higher compared to a client in the current Internet, but we presented some arguments for sending at higher rates in Section 4.2.4.

Figure 5.3.2 shows the setup delays experienced by a legitimate client at different attacker rates. On average, the setup delay is below 10 seconds, while the error bars show that in some cases the setup delay may be longer. For instance, the setup delay is close to 50 seconds when the aggregate attacker rate is 120 rps. However, the figure also shows the number of failed transactions (out of 50 trails) for each attacker rate; as the

attack rate increases the number of failures also increases, but dramatically. For instance, at an aggregate attack rate of 300, although the setup delay is close to 10 seconds, the legitimate client fails to reach the destination 30 out of the 50 trails. In other words, the legitimate clients can sustain a relatively high success rate even when the attackers are flooding at 5 times the request rate-limit capacity. Not only that, but when the legitimate client's request retransmission rate is high, the legitimate client can expect to succeed within 10 seconds after it initiates a transaction.

Figure 5.3.2 also shows the data transfer delay. The average data transfer delay is 10 seconds, which is understandable, since the legitimate client has 1000 data packets to transfer and the data transfer rate limit is 100, the client should take approximately $\frac{1000pkts}{100dps} = 10$ seconds to transfer the data. However, the error bars show that in some cases, the data transfer delays were over a minute. For instance, at an attacking rate of 90, the maximum data transfer time for the legitimate client was close to 70 seconds. On careful inspection of the data logs, we found that the destination process running on a PlanetLab host stalled for nearly 60 seconds in total; the stall causes the destination to not send acknowledgements to data packets it receives, which in turn causes the client to stall. We found similar events for the other cases as well. Although we are unable to pinpoint the exact cause of this problem at this time, we strongly believe that it is caused due to the very high load on PlanetLab hosts and an effect of virtualization process scheduling. For instance, most PlanetLab hosts we used for the experiments have anywhere from 40 to 60 active slices (virtual machines) running on them at any time. The experiment clearly shows that data transfer time is not affected by the presence of attackers in Epiphany, since the legitimate client uses a different proxy to transfer the data. In comparison, Figure 5.3.3 shows the case when a single proxy acts as both an SP and a DP. Here, the destination sets a single rate limit of 50 packets per second at the proxy. The proxy accepts both requests and data packets and forwards them

to the destination, but does not exceed the rate-limit. In other words, if a legitimate client is attempting to send 1000 data packets through this proxy, it will take $\frac{1000}{50} = 20$ seconds on average if there are no attackers present. However, as the attackers send more requests, the proxy will start to drop requests and data packets indiscriminately and the data transfer delay for the legitimate client increases. For example, in Figure 5.3.3 when the attackers are sending at 30 requests per second, the data transfer delay takes a little over 20 seconds on average (out of 50 trials), but as the attacking rate increases, the data transfer delay also increases dramatically.

In summary, separating the proxies helps prevent collateral damage to legitimate traffic in the presence of attackers, while using the same proxy for both data transfer and setup causes collateral damage.

# Chapter 6

# Conclusions & Future Work

DDoS defenses are generally hard to deploy considering the changes they require to the network infrastructure. We analyze the changes Epiphany makes and layout a plan for deploying Epiphany in Section 6.1. We then provide a summary and describe the important contributions and impact of our work in Section 6.2. Epiphany is a new architecture and there are a few directions we wish to explore in the future. We describe these in Section 6.3.

## 6.1 Deployment

Deploying Epiphany on the Internet requires some new components such as the proxies and making changes to the software of existing components such as routers and hosts. We now describe the changes and the deployment strategy for each of these components.

### 6.1.1 Proxies

Epiphany relies on large number of proxies for its defenses. As a starting point, the Epiphany data proxies could be recruited from content distribution networks such as Akamai [1], which currently has roughly 95,000 servers spread in 1,900 networks. Unlike replicated services, Epiphany data proxies are light-weight; they do not hold large databases of user account information or require special hardware, and they operate independently from other proxies (they do not form proxy networks as in SOS [17], or TOR [12]). All the state that they maintain is generated during a session with the destination. If a DP is attacked, it can be removed/replaced and brought back when it is no longer under attack. In other words, DPs can be implemented alongside of existing

replicated instances on Akamai CDN nodes. However, note that with this approach the attacker will know about the DPs and could potentially disrupt communication of important clients. One approach to solving this is to use a private set of nodes along side of the publicly know Akamai nodes. The destination could assign untrusted or less important clients to the Akamai nodes and given the private DPs to more important clients.

Setup proxies in Epiphany require additional support from the routing infrastructure to implement anycast. Although anycast is widely used to protect top level DNS servers, anycast introduces additional routes (and routing updates) into the global Internet routing and hence must be used prudently. We outline two mechanisms to relieve this stress on global routing. In the first approach, we propose to use a combination of locally scoped and globally scoped anycast SPs, as shown in Figure 6.1.1. Networks A and C have hosts they wish to use as SPs, and the destination is willing to recruit them for the SP. On the other hand, networks B and D either do not have hosts to offer as SPs or the destination is not willing to install SPs in their networks. The anycast addresses for the SPs in networks A and C are announced within these networks and their announcements are locally scoped. That is, before networks A and C export any internal routes to the global routing space, they remove the route for the SP. Since the routes for these SPs are local, they do not create additional routing state in global routing, or the failure of any of these SPs does not cause a route change in global routing. Networks B and D must use the globally scoped anycast SPs to reach the destination. To reduce the number of routing entries, the SPs could share a common address prefix with other anycast services. For instance, several Epiphany hidden services could share a common address prefix (say a /24) and announce that prefix to global routing; each hidden service will then use a different address from this space for its SP.

An alternative is to apply the techniques described in [9, 16]. In this approach, a few anycast SPs are globally scoped (with possibly a shared address prefix as described

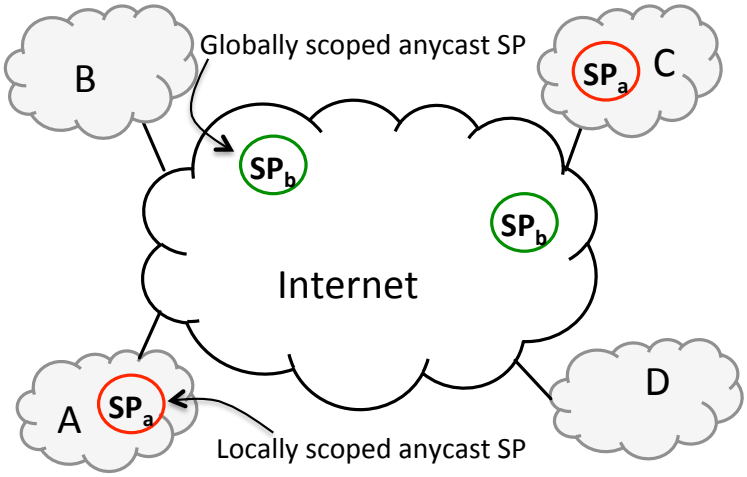


**Figure 6.1.1:** A deployment strategy for Epiphany Setup Proxies

above). However, clients are not allowed to use these SPs directly. Instead, the SPs only accept requests from a specific set of hosts that are part of a peer-to-peer (p2p) type network. The clients send their requests to the p2p hosts and these hosts forward them to the SP. The p2p hosts can also perform rate-limiting before sending the requests to the SPs. Since the attacker may attempt to flood the SPs directly without going through the p2p network, the SP network must employ some form of filtering at its border routers to eliminate the flood. The advantage of this approach is that, first, large number of SPs are possible due to the large scale of end hosts based p2p networks and second, the destination does not need to construct hidden paths to a large number of SPs and hence less overhead on the destination and on the routers.

### 6.1.2 Routers

Constructing hidden paths on routers consumes forwarding state, which is a premium on some Internet routers. To reduce the state, cryptographic based path generation techniques similar to SNAPP [33] may be employed. However, these techniques require making changes to the packet structure and may additionally require making hardware changes to routers (to support high speed cryptographic operations). An alternative is

to use tunneling where it is necessary, and the state based approach where possible. For instance, an intermediate router inside a network could use an MPLS tunnel to forward the packets to a border router that has an Epiphany forwarding entry. In addition, since reaching the service is more important than efficiency, routers could use slower (but more abundant) memory for forwarding Epiphany packets rather than high speed memory.

As for the software changes on routers to implement the new protocol for Epiphany hidden paths, our implementation on Cisco 2600 routers shows that single source multicast could be used as an alternative until the full Epiphany protocol is implemented on routers. Moreover, since many functions in Epiphany are similar to multicast routing (such as the multicast forwarding table), the implementation could borrow these existing functions to implement Epiphany hidden paths. Also, note that Epiphany does not make changes to the IP forwarding engine itself, so those components remain unchanged. The required changes could be applied incrementally, while resorting to tunneling on non-Epiphany aware routers.

### 6.1.3 Hosts

In order for a source to communicate with the destination it needs to, 1) send a request to a host but specify a different service name, and 2) use a different host to retrieve the actual resource. At the basic level, applications could be modified to support this model while the host software remains unchanged. However, protocols such as HTTP already support these features (albeit without the added security we specify). For instance, an HTTP client sends can send an HTTP GET request to the SP, but specify the destination's service identifier in the HOST field of the request. The SP needs to be able to parse the HTTP request, extract the destination's identifier and forward the request to the destination. To this end, we could make the SP HTTP aware. When the destination receives the request, it generate an HTTP redirect message with the DP as the

redirected host and sends it to the SP. The SP simply sends the redirect message to the client, which then learns about the DP and switches to the DP to reach the destination. Thus, a service that operates on HTTP can simply use the HTTP redirect mechanism to implement Epiphany without making modifications to the client. Applications that are not based on this model, such as FTP, SSH and so forth on the other hand require some software changes. These changes could be implemented incrementally as more and more applications move to Epiphany. Note that the HTTP protocol does not support additional encryption to protect the redirect mechanism. The DP address (i.e., redirected host address) is sent in the clear through the SP. If the SP is compromised or malicious, it will reveal the DP's address to the attacker.

## 6.2 Summary

The tremendous growth of the Internet and its allure for increased flexibility and availability has led many critical services to migrate to the Internet, such as emergency response services, government portals for national security, military and diplomatic operations, power distribution and industry control systems, banks and so forth. However, DDoS attacks pose significant threat to these services and challenge the very availability aspect of the services.

In this dissertation, we propose a novel location hiding architecture to protect critical services from DDoS attacks. Our architecture achieves strong location hiding property; neither the sources nor the proxies know the destination address. Moreover, the destination network itself is not announced into global routing, and therefore nobody can send packets to the hosts inside that network without going through the Epiphany communication model. Since the attacker does not know the destination address, directed attacks on the service are no longer possible. We achieve this location hiding using two important components, proxies and hidden paths. Epiphany hidden paths share many

similarities with multicast routing and can be implemented on routers without sacrificing forwarding efficiency, or requiring changes to the router architecture. Hidden paths operate on existing IP packet and address structure, and the multicast functions for forwarding and membership management available on routers could be reused to construct them.

Like other location hiding approaches, Epiphany employs large number of proxies for its defenses, but instead of making them public Epiphany separates them into setup and data proxies. Only the setup proxies are made public while the data proxies are hidden and revealed to important/trusted clients. This is an important property of Epiphany; since the data proxies are not public, the attacker cannot disrupt established communications of other legitimate clients in the network unless the attacker knows about the data proxy a specific client is using. We demonstrate this property of Epiphany using an overlay based implementation of Epiphany on Planetlab hosts. Our evaluation shows that making the proxies public allows the attacker to consume their resources and make the data transfer times increasingly longer with higher attacker rates, while in Epiphany the attack on setup proxies has no effect on established communications.

The next important property of Epiphany is the use of IP anycast to localize attackers. Since the setup proxies are public, the attacker could amass a large botnet and target them. Epiphany has an open communication model, i.e., any source can send a request through the SP and ask the destination for further access. The setup proxies do not treat different clients preferentially and therefore let any request go through. To avoid request flooding attacks, the destination assigns rate limits to the SPs. Now, if the SPs are all accessible, the attacker could simply consume the request resources at each SP and prevent legitimate clients from reaching the destination, or in other words, cause DoC attacks. Network capability architectures have a similar problem, but address them using cryptographic puzzles based approach [34]. In Epiphany, we address these attacks

using a two pronged approach. First, Epiphany SPs use anycast addresses, which creates distinct regions in the network. Requests generated from clients in one region are automatically forwarded to the SP in that region. Therefore attackers become dispersed and cannot prevent legitimate clients in regions that have no attackers from reaching the destination. Additional anycast SPs could be brought up in regions that have a large attacker presence to further split them up and localize them to smaller and smaller regions. Second, once the attackers become localized, they can either be located and removed, or simple resource proof approaches such as client puzzles [43] could be used to limit the damage they can cause. We show the effectiveness of anycast SPs using simulations on an Internet scale topology. Our analysis found that for a small legitimate client density of 0.01%, an SP density of 0.5% is sufficient to suppress over 93% of the attackers even when 10% of the entire network is occupied by attackers. Moreover, at that density the attackers are localized to within 3 to 4 hops of the SPs, and most often there is only attacker interfering with legitimate clients at an SP. Also, placing the SPs closer to the edge networks, i.e., where the attackers and legitimate clients are situated is the best strategy for SP placement when there are a large number of SPs, while placing them so that there is at least one SP per AS is a better strategy when there are far fewer SPs.

Finally, the attackers may compromise proxies and funnel DDoS traffic through them to cause collateral damage to legitimate traffic. Once the destination detects such a misbehaving proxy, it can immediately remove the hidden path to that proxy to prevent it from causing further damage. Moreover, the destination could quickly add new proxies to handle large scale attacks. To test these properties we constructed a small scale Epiphany tesbed on Cisco routers and used single source multicast to build the hidden paths. In our measurements, we found that building a hidden path takes only a fraction of a second, but tearing it down takes in the order of seconds. Although in the Internet, factors such as propagation delay and processing delay at routers will add to the overall

delay, we believe they will still be within order of seconds. In other words, the damage that a compromised proxy can cause is limited to within seconds.

In summary, we were successful in developing a strong location hiding architecture with the specific goals we set forth for the dissertation. We believe our architecture is a new dimension in the overall DDoS defenses that are based on location hiding. Although prior techniques have used location hiding principles, to the best of our knowledge none of them have completely hidden the destination. Moreover, using well established techniques, we were able to develop a proxy system that is highly localized and involves low security throw away boxes. We neither rely on or have trusted proxies. Finally, we separate the proxies and allow the destination to decide how it will assign proxies, and to whom it will assign those proxies. Through the deployment strategy we outlined above, we believe Epiphany can be deployed effectively in the Internet in the near future.

## 6.3 Future Work

We believe there are a few directions we could explore in the future. For instance, just as any other DDoS defense architecture Epiphany makes a few design tradeoffs, and also makes assumptions about specific attacks. We briefly describe these here and leave the detailed analysis for future.

First, Epiphany uses a state based approach to construct hidden paths, but does so without requiring changes to the IP packet structure or forwarding functions at routers. A different approach, loosely based on [33] is to trade the per-destination state for cryptographic operations, albeit with the loss of compatibility with IP packet structure and forwarding functions at routers. To sketch a rough design, consider that when a router receives an rjoin, instead of creating an EFIB entry, the router uses a locally generated cryptographic key to encrypt the interface identifier on which it received the rjoin. The router then inserts the encrypted token into the next available location in a variable

length source address field of the rjoin message, and forwards the rjoin to its upstream router on the path to the proxy. Each router on the path similarly insert an encrypted form of their interfaces on which they received the rjoin. When a proxy receives the rjoin, it extracts the source address field, which is now an encoded route to the destination and creates an entry for the destination in its local Epiphany table. When a packet arrives for the destination, the proxy inserts the encoded route into a variable length destination address field of the packet and sends it to the attached router. The router extracts its own portion of the encoded route and decrypts it to find out the outgoing interface on which it should send the packet. Each router independently decrypt the route and forward the packet to the destination. The benefit of this approach is that it does not consume any state to create or maintain the hidden path at routers (except for the state required to store the cryptographic key). Moreover, since the encoded path is included in every packet that a proxy sends, the destination can easily identify misbehaving proxies if they attempt to flood the destination. Unlike the path identifier based approach we discussed in the design section, the encode path is unique to every proxy and hence the accuracy of detection is very high. However, as noted earlier, the cryptographic based approach changes the packet structure and forwarding functions at routers and therefore deployment may be difficult. We believe these are compelling but opposing design choices, and a detailed analysis is important to understand the tradeoffs.

Another design tradeoff is the need to deploy a large number of SPs versus the overhead in constructing hidden paths to all the SPs. Hidden paths require committing state resources at routers and also require making changes to the router software. Initially not all routers may have implemented these changes and hence deploying a large number of proxies close to the edges may not be possible. In addition, anycast route propagation has some inherent delays depending on the underlying routing protocol in use, which also affects how quickly the destination can react to attacks. In the deployment section

we discussed an alternate design in which the destination appoints a few concentrator SPs that accept and forward requests from a p2p based SP network. The p2p network of proxies do not use anycast addresses, and the destination does not need to construct hidden paths to each and individual SP. However, such a design has numerous issues involving management, security and proxy discover for the clients. We wish to explore these issues and consider the solution as a future design element in Epiphany.

We also wish to further study some of the attacks we outlined in the design. Specifically, 1) understand how the system behaves when the attacker games the destination into revealing data proxies. If the destination's policy is to assign a few data proxies to new clients, the attacker could learn about all these DPs (by asking the destination using new compromised clients) and prevent other legitimate clients that are also classified as new clients from reaching the destination. 2) The attackers could flood the network with hidden path rjoin messages and can consume router state resources. To prevent it, we described configuration at ISPs. However, it is possible that not all networks are configured to reject messages from attackers. Further analysis is needed to understand under what circumstances such attacks are possible. The findings could be used to direct new designs to prevent such attacks.

Finally, we wish to expand our anycast evaluation using specific attacker, SP and legitimate client placement strategies. The results we reported in the dissertation assume a simple random distribution. However, in the real Internet, the location of each of the players affects how many attackers are rendered ineffective and how many legitimate clients are free from attacker. Knowing these numbers will greatly enhance the understanding about the effectiveness of our system. The current CAIDA topology data does not include end host information, but perhaps future data may include such information, allowing us to map known attacker bot locations and legitimate client locations in the topology. Then, we may try different SP placement strategies and study their impact.

# REFERENCES


[1] Akamai technologies. `http://www.akamai.com`.

[2] Click modular router. `http://read.cs.ucla.edu/click/`.

[3] Planetlab. `http://www.planet-lab.org/`.

[4] J. Abley and K. Lindqvist. Operation of Anycast Services. RFC 4786, December 2006.

[5] David G. Andersen. Mayday: Distributed filtering for Internet services. In *USENIX USITS*, 2003.

[6] Tom Anderson, Timothy Roscoe, and David Wetherall. Preventing Internet denial-of-service with capabilities. *SIGCOMM Hot Topics in Networks (HotNets-II)*, November 2003.

[7] Katerina Argyraki and David Cheriton. Network capabilities: the good, the bad and the ugly. In *HotNets-IV*, November 2005.

[8] Katerina Argyraki and David R. Cheriton. Active Internet traffic filtering: Real time response to denial-of-service attacks. In *USENIX*, 2005.

[9] Hitesh Ballani and Paul Francis. Towards a global IP anycast service. *SIGCOMM*, 2005.

[10] CAIDA. The internet topology data kit – 2011-04.

[11] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, October 2002. Updated by RFC 4604.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX*, 2004.

[13] Colin Dixon, Thomas Anderson, and Arvind Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In *NSDI*, 2008.

[14] H. Holbrook and B. Cain. Source-Specific Multicast for IP. RFC 4607, August 2006.

[15] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Network and Distributed System Security (NDSS'02)*, 2002.

[16] Dina Katabi and John Wroclawski. A framework for scalable global IP-anycast (GIA). *SIGCOMM*, 2000.

[17] Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. SOS: Secure overlay services. In *ACM SIGCOMM*, pages 61–72, August 2002.

[18] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. Taming IP packet flooding attacks. *SIGCOMM CCR*, 34(1):45–50, January 2004.

[19] Xin Liu, Xiaowei Yang, and Yanbin Lu. To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *ACM SIGCOMM*, pages 195–206, 2008.

[20] Robert Mackey. 'operation payback' attacks target mastercard and paypal sites to avenge wikileaks. New York Time Blog.

[21] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM CCR*, 32(3):62–73, July 2002.

[22] Allison Mankin, Dan Massey, Chien-Lung Wu, S. Felix Wu, and Lixia Zhang. On design and evaluation of "intension-driven" ICMP traceback. In *Computer Communications and Networks*, pages 159–165, 2001.

[23] Danny McPherson and Dave Oran. Architectural considerations of IP anycast. Draft-iab-anycast-arch-implications, February 2010.

[24] Jelena Mirkovic and Peter Reiher. Building accountability into the future internet. In *NPSec*, pages 45–51, October 2008.

[25] Ryan Naraine. ICANN: Anycast saved DNS root servers during attack. "`http://blogs.zdnet.com/security/?p=118`".

[26] Maitreya Natu and Jelena Mirkovic. Fine-grained capabilities for flooding DDoS defense using client reputations. In *Workshop on Large Scale Attack Defense (LSAD)*, pages 105–112, 2007.

[27] Jose Nazario. Estonian DDoS attacks. `http://ddos.arbornetworks.com/2007/05/estonian-ddos-attacks-a-summary-to-date/`.

[28] Jose Nazario. Georgia on my mind – political DDoS. `http://asert.arbornetworks.com/2008/07/georgia-on-my-mind-political-ddos/`.

[29] Arbor Networks. Worldwide infrastructure security report – Volume VI. `http://www.arbornetworks.com/report`, 2010.

[30] Lasse Overlier and Paul Syverson. Locating hidden servers. *IEEE Symposium on Security and Privacy*, 100-114, 2006.

[31] Lasse Øverlier and Paul Syverson. Valet services: Improving hidden servers with a personal touch. *Workshop on Privacy Enhancing Technologies*, 2006.

[32] Lasse Øverlier and Paul Syverson. Location hidden services and valet nodes. *Privacy in Telecommunications, Telektronikk*, 2007.

[33] Bryan Parno, Adrian Perrig, and Dave Andersen. SNAPP: Stateless network-authenticated path pinning. In *ASIACCS*, pages 168–178, 2008.

[34] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *ACM SIGCOMM*, August 2007.

[35] Collins M. Patrick, Shimeall Timothy J., Faber Sidney, Janies Jeff, Weaver Rhiannon, De Shon Markus, and Kadane Joseph. Using uncleanliness to predict future botnet addresses. *IMC*, 2007.

[36] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for IP traceback. In *ACM SIGCOMM*, pages 295–306, August 2000.

[37] Neil Spring, Larry Peterson, Andy Bavier, and Vivek Pai. Using PlanetLab for network research: Myths, realities, and best practices. *SIGOPS*, 2006.

[38] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *IEEE/ACM Transaction On Networking*, pages 205–218, 2004.

[39] UltraDNS. Siteprotect system. `http://www.ultradns.com/ddos-protection/siteprotect/how-it-works`.

[40] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. Ddos defense by offense. In *ACM SIGCOMM*, pages 303–314, August 2006.

[41] Ju Wang and Andrew A. Chien. Understanding when location-hiding using overlay networks is feasible. *Computer Networks*, 2006.

[42] Lan Wang, Qishi Wu, and Dung Dinh Luong. Engaging edge networks in preventing and mitigating undesirable network traffic. In *Workshop on Secure Network Protocols (NPSEC)*, October 2007.

[43] Xiaofeng Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *IEEE Symposium on Security and Privacy*, pages 78–92, May 2003.

[44] Xiaofeng Wang and Michael K. Reiter. Mitigating Bandwidth-Exhaustion Attacks using Congestion Puzzles. In *ACM Computer and Communications Secutiry (CCS'04)*, October 2004.

[45] Abraham Yaar, Adrian Perrig, and Dawn Song. Pi: A path identification mechanism to defend against DDoS attacks. In *IEEE Symposium on Security and Privacy*, pages 93–107, May 2003.

[46] Abraham Yaar, Adrian Perrig, and Dawn Song. Siff: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy*, pages 130–143, May 2004.

[47] Abraham Yaar, Adrian Perrig, and Dawn Song. Fit: Fast Internet traceback. In *IEEE INFOCOM*, volume 2, pages 1395–1406, 2005.

[48] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-limiting network architecture. In *ACM SIGCOMM*, pages 241–252, August 2005.