Carnegie Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy

TITLE **Toward a Highly Available Future Internet**

PRESENTED BY **Hsu-Chun Hsiao**

ACCEPTED BY THE DEPARTMENT OF

Electrical and Computer Engineering

Adrian Perrig	6/5/2014
ADVISOR, MAJOR PROFESSOR	DATE
Jelena Kovacevic	6/5/2014
DEPARTMENT HEAD	DATE

APPROVED BY THE COLLEGE COUNCIL

Vijayakumar Bhagavatula	6/5/14
DEAN	DATE

Toward a Highly Available Future Internet

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Hsu-Chun Hsiao

B.S., Electrical Engineering, National Taiwan University
M.S., Electrical Engineering, National Taiwan University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

June 2014

This research was supported by CyLab at Carnegie Mellon University (CMU), and by support from National Science Foundation (NSF) under awards CCF-0424422 and CNS-1040801. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of CMU, NSF or the U.S. Government or any of its agencies.

Abstract

Numerous cyberattack incidents have demonstrated adversaries' capability to cause Internet outages lasting hours or even days, jeopardizing governmental, financial, telecommunication, transportation, and healthcare services. With the intensity and frequency of cyberattacks constantly growing, it becomes crucial to find a way to provide availability guarantees despite active adversaries.

As an initial step toward building a highly available Internet, this dissertation explores how to secure the data plane against off-path adversaries that flood the network (Distributed Denial of Service attacks) as well as on-path adversaries that discriminate against traffic (selective dropping attacks). These two types of attacks are increasingly prevalent, and their mitigation will lead to substantial improvement in Internet availability. However, DDoS and selective dropping attacks have yet to be addressed efficiently and effectively in the current Internet, primarily due to the fact that many security problems are too pervasive and fundamental to be fixed using patches constrained by the underlying Internet architecture.

To address this challenge, this dissertation studies complementary defense mechanisms on top of a recently proposed clean-slate Internet architecture. This new architecture is a key enabler of this dissertation, as it provides several useful architectural primitives to support fine-grained isolation and fair access to resources—two driving principles behind the design of the defense mechanisms in this work. Fine-grained isolation protects legitimate traffic from interference with other traffic, including attack traffic. When isolation is not possible, fair access guarantees that legitimate traffic receives a fair share of resources during resource competition.

Guided by these two principles, the first part of the dissertation describes novel solutions that cover three aspects of DDoS defense: (1) isolating DDoS traffic via bandwidth reservation, (2) bounding the waiting time before a successful reservation of flow bandwidth, and (3) ensuring that every flow complies with its allocated bandwidth limit without keeping per-flow state. The second part of the dissertation explores the prevention of address-based selective dropping through topological anonymity, and describes a lightweight anonymous forwarding scheme with near-optimal latency under a relaxed attacker model. The integration of these mechanisms achieves end-to-end availability guarantees on top of the clean-slate architecture under the assumptions that the lower layers are available and the routing paths are given. The resulting guarantees are independent of the strength of remote attackers, a feature that none of the existing work is able to achieve. Since several of the proposed mechanisms can be deployed incrementally for incremental protection, they can also improve the availability of the current Internet.

Acknowledgments

I would like to express my deepest appreciation to each and every person who helped me throughout my Ph.D. journey.

First and foremost, I would like to thank my advisor, Dr. Adrian Perrig, for his continuous guidance, inspiration, and encouragement. His enthusiasm about research and his confidence in my ability to overcome challenges motivated me to live up to my full potential whenever I was in doubt. I would also like to thank my thesis committee, Dr. Virgil Gligor, Dr. Yih-Chun Hu, and Dr. Srinivasan Seshan, for their insightful comments and suggestions at various stages of this work. Discussions and conversations with them have been invaluable and have tremendously broadened my views as a researcher.

This dissertation would not have been completed without the extensive assistance from my committee members as well as many other collaborators, to whom I am deeply indebted: Dr. Hyun Jin Kim, Dr. Soo Bum Lee, Dr. Xin Zhang, Tae-Ho Lee, Dr. Yue-Hsun Lin, Dr. Marco Gruteser, Hao Wu, Dr. Akira Yamada, Sangjae Yoo, Wei Meng, and Dr. Samuel Nelson. In my early research projects, I was extremely fortunate to have worked with many talented researchers and students. In particular, I would like to thank Dr. Ahren Studer, whose patience and guidance were essential for the completion of many of these early projects.

I am also grateful to the faculty members, staff members, and students at Carnegie Mellon University for creating an enjoyable working environment. I am particularly thankful to Dr. Limin Jia, Dr. Lujo Bauer, Dr. Maverick Woo, Yanlin Li, Dr. Edward Schwartz, and Dr. Zongwei Zhou, who always offered me excellent advice in both life and research.

My journey as a Ph.D. student was made possible thanks to the inspiration and support I received from Dr. Joseph Hellerstein, Dr. Chin-Laung Lei, and Dr. Doug Tygar. During this journey, I also had opportunities to interact with many wonderful people at UC Berkeley, iCast, Qualcomm, IBM Zurich, and ETH. These interactions enriched many aspects of my life, for which I am truly grateful.

The enduring support provided by my caring friends and family allowed me to stay strong in my search for myself. Thanks to Hyun Jin for being a supportive friend and reliable life mentor. Thanks to Wei for sharing everything, including ideas, beers, and optimism. To my parents, sisters, and grandparents, I cannot thank you enough for your belief in me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the world continues to become more digitized and interconnected, the availability of the Internet becomes increasingly valuable. Disruptions to the Internet infrastructure and communications can cause tremendous harm to us in many aspects of our lives. Indeed, the World Economic Forum ranked "critical information infrastructure breakdown" as one of the top five global risks in terms of impact in 2014 [166]. The examples below illustrate the impact that these disruptions can have on individuals, societies, and economies:

- **User frustration and low productivity.** The number of Internet users grew from 6 million to 2.7 billion within the last decade (from 2003 to 2013). Users spend an average of 6.9 hours per day on the Internet via desktop, laptop, and mobile devices [5] for email, instant messaging, social networking, gaming, audio/video streaming, file sharing, web browsing, etc. Being unable to access Internet-based services not only causes widespread frustration but also hinders productivity.

- **Loss of revenue for financial services and online retailers.** Amazon found that its sales dropped by 1% for every 100ms increase in page load time [88]. In August 2013, Amazon lost about two million dollars for a thirty-minute outage [34]. In another instance, trading firms lost approximately 500 million dollars when the Nasdaq stock exchange was down for three hours in 2013 [142].

- **Potential loss of life for users relying on critical Internet-based infrastructures.** Telemedicine, smart grids, and connected vehicles are becoming a reality in the "Internet of Things" era. As more and more critical infrastructures are overlaid on top of the Internet for lower-cost communication, Internet outages may put human lives at risk. For example, in the face of communication disruption or delay, real-time alarm systems such as medical monitors and smoke detectors may fail to alert emergency responders in time. In telesurgery, similarly, where distant surgeons control surgical robots over communications networks [98], Internet outages may create life-threatening situations.

Given these risks, ensuring high availability is of paramount concern for governmental, financial, telecommunication, transportation, and healthcare services. Specifically, there is a need for an Internet that allows any client to access any service at any time and that provides actionable and meaningful availability guarantees in terms of waiting time and bandwidth in an efficient manner despite active adversaries. Unfortunately, end-to-end communication in the current Internet lacks these guarantees, as communication may be interrupted unpredictably for various reasons, ranging from natural disasters to unintended failures to malicious attacks.

Section 1.1 examines the major causes behind disruptions and outages in the current Internet. Section 1.2 discusses the challenges of building a highly available Internet despite active attacks. We then state our thesis statement in Section 1.3, followed by the thesis overview in Section 1.4 and the thesis outline in Section 1.5.

1.1 Threats against Internet Availability

The Internet is a collection of interconnected networks. By connecting to the Internet, endhosts can access any Internet-based services and communicate with each other. However, even though an endhost can successfully access the Internet, the Internet itself may fail to reliably deliver packets from one end to another due to diverse threats against its architecture or infrastructure.

Threat taxonomy. Threats against Internet availability can be broadly classified into three types[1]:

1. **Natural disasters** are threats outside human control, such as earthquakes, floods, storms, etc. In 2006, an earthquake in Taiwan and the subsequent submarine landslides severely impaired seven out of nine geographically co-located cables in the Luzon Strait, resulting in a six-hour outage for more than two thousand IP prefixes [97, 133]. During Hurricane Sandy in 2012, the Internet outage rate in the U.S. was doubled, and the impact lasted for about four days [73]. From a technical perspective, the best protection against this type of threat is ensuring high resilience against natural disasters through redundancy and replication and supporting rapid recovery after disasters occur.

2. **Unintended failures**, such as misconfiguration and buggy code, can often be eliminated through better administration. Incidents of network failures (e.g., damaged cables or misconfigured routers)

[1]We adopt the root cause categories used by the European Union Agency for Network and Information Security [156] and consider human errors, system failures, and third party failures to all be forms of unintended failures.

can result in hours of connection disruptions before recovery or rerouting. In 2007 and 2011, buggy routers from large router vendors falsely reconstructed their routing tables causing widespread network outages in Japan [52] and the U.S. [53]. Sometimes the technical problems cannot be easily fixed by individual administrative organizations. For example, unstable routes, which may persist long after a route change due to delayed convergence, severely degrade the performance of end-to-end communications; measurements show that up to 30% of packets are lost within two minutes of a BGP route update [94].

3. **Malicious attacks** are intentional attempts to disrupt network communication. Attacks against availability can occur in many forms, ranging from the exploitation of application-specific vulnerabilities to the interference with transmission at lower protocol layers. Generally, these attacks can target any logical layer/component or any physical device that is required for end-to-end communication. For example, prefix hijacking attacks target the routing protocol at the network layer; DNS poisoning attacks target the Domain Name System at the application layer; and Distributed Denial of Service (DDoS) attacks overwhelm the network infrastructure or servers with a high volume of traffic in an attempt to crowd out legitimate traffic. Successful end-to-end communication heavily depends on the availability and security of each of these components and devices.

We acknowledge that a complete solution of Internet availability requires the protection of all layers against all threats. As malicious attacks are expected to grow in both power and sophistication since attackers will also benefit from these advances in technology, this dissertation focuses on one challenging aspect of the problem: the mitigation of emerging data-plane attacks at the network layer.

Major data-plane attacks. The data plane is responsible for forwarding packets along a given path to the intended destination, assuming the path exists and is discovered by the underlying routing protocol. Despite substantial research efforts on securing the Internet, two data-plane attacks continue to pose serious challenges that have not yet been effectively or efficiently resolved. *Distributed Denial of Service (DDoS)* attacks are *off-path data-plane attacks* that disrupt communication by sending excessive traffic exhausting bottleneck resources, such as network link bandwidth and server CPU cycles. *Selective dropping* attacks are *on-path data-plane attacks* that disrupt communication by directly blocking a certain type of traffic. These two types of data-plane attacks are easy to launch yet hard to defend against, leading to their increased prevalence. The severity and prevalence of these two types of attacks is evident from many real-world incidents, as summarized below:

- **DDoS attacks.** A DDoS attack often involves an attacker controlling a large number of compromised machines that send large traffic volumes to overwhelm the Internet infrastructure or servers. DDoS attacks have become a common means for various purposes, ranging from political protest to extortion, and have malevolently paralyzed many critical services. For instance, in 2010, a hacktivist group named Anonymous attacked sites including PayPal, Visa, and MasterCard to protest against these companies as they stopped accepting donations to WikiLeaks. In 2012 and 2013, a large number of banking sites were down for hours due to DDoS. Both the intensity and frequency of these attacks have drastically accelerated. A survey in 2012 showed that the number of attacks increased by 25% compared with the previous year [134], and the largest attack to date reached 300 Gbps [115]. Worse yet, as DDoS attackers get smarter and try to mimic flash crowds, it becomes extremely difficult to distinguish DDoS traffic from normal traffic based on behavior signatures.

- **Selective dropping attacks.** In selective dropping, an attacker controlling devices in the Internet selectively blocks critical communication based on certain criteria, such as contents, network addresses, etc. Government censorship and data discrimination by Internet Service Providers are both in this category. The recent revelation of large-scale surveillance by the U.S. National Security Agency has raised serious concerns about government surveillance and censorship. In 2013, Reporters Without Borders identified Syria, China, Iran, Bahrain, and Vietnam as countries whose governments are actively involved in censorship and surveillance [140]. The OpenNet Initiative has documented government censorship in over forty countries [127]. Besides government-level selective dropping, ISPs are known to have a history of discriminating against rival services and throttling bandwidth-demanding services. Evidence was found confirming traffic discrimination in backbone ISPs [182]. In early 2014, Verizon's customers experienced slowdowns in watching Netflix videos, and there was suspicion that Verizon was deliberately throttling Netflix's traffic [25]. Since content-based selective dropping can be mitigated by encrypting the payload using end-to-end encryption, this work focuses on address-based selective dropping, where the attacker blocks packets to and/or from some specific hosts.

1.2 Challenges of Building a Highly Available Internet

A desirable goal is to build a highly available Internet in which the source can send packets to the destination with a high probability of success, even in the face of DDoS and address-based selective dropping adversaries. From the user's perspective, the desire would be the obtainment of non-trivial availability guarantees for data delivery in an efficient manner.

In this thesis, we focus on two primary dimensions of end-to-end availability for the communication between two entities in the Internet: *waiting time* and *bandwidth*. Specifically, our goal is to provide:

1. **Waiting time guarantees**, which ensure an upper bound on the time for a user to access a service. Waiting time bounds are important, since various studies suggest that Internet users are impatient to wait and prefer to know how long they need to wait in advance [3, 113, 124, 24]. Note that waiting time includes the time to retry due to failed delivery attempts. Waiting time is different from end-to-end latency that considers the time taken for a packet to be transmitted across a network from the source to the destination.

2. **Bandwidth guarantees**, which ensure that (1) the sender can obtain a lower bound on the reservable bandwidth for a flow, and (2) once reserved, the sender is guaranteed to have the reserved amount of bandwidth regardless of any other traffic.

A flow contains packets sharing the same source and destination, and there can be multiple flows between a pair of endhosts. It is up to the source and destination endhosts to divide traffic among flows.

Note that most prior work on bandwidth reservation (e.g., RSVP [177]) does not address security aspects, and is thus unable to offer such guarantees in the presence of adversaries. Without any security countermeasures, reservation requests may be attacked and prevented from reaching the destination (i.e., no waiting time guarantees). Even when a request successfully arrives at the destination, the reservable amount of bandwidth along the route may be too low for any real use case (i.e., no guarantees on reservable bandwidth); finally, without proper monitoring and regulation, the reserved slots may still be malevolently occupied by malicious senders who use more than their allocated amounts (i.e., no guarantees on the reserved bandwidth).

We seek to provide strong guarantees that are deterministic rather than probabilistic and that have little or no dependency on the adversary's power. In addition, such guarantees should be *meaningful* and *actionable*. In the context of improving Internet availability despite attacks, meaningful guarantees help to mitigate attacks and improve service quality in terms of waiting time and bandwidth, while actionable guarantees (or enforceable accountability) enable users to take legal or contractual actions when their providers fail to meet their promises. Guaranteeing that a packet will eventually arrive at the destination may not be meaningful to users, as it does not improve much from the current Internet. Meaningful and actionable availability guarantees are important as they can promote new business models such as availability-as-a-service and in turn incentivize Internet stakeholders to evolve toward a highly available Internet.

However, this goal is challenging and has not yet been accomplished by *patching the current Internet* for three main reasons:

1. **Current Internet protocols and architecture mainly focus on reliability perspectives, rather than on security.** A reliable system should withstand failures (e.g., hardware/software bugs and configuration errors), and a secure system should withstand external attacks. For example, TCP is reliable but not secure. While TCP is designed to reliably handle packet loss using re-transmission, it cannot prevent malicious routers from deliberately blocking all packets.

2. **Existing security mechanisms are either inefficient or ineffective against strong attacks.** Existing security mechanisms often incur high overhead in peacetime and become ineffective in the face of strong attacks, such as the Coremelt DDoS attack [149], which cannot be remedied by any DDoS defense work to date.

3. **Security problems are pervasive, fundamental, and hard to fix using patches constrained by the underlying Internet architecture.** As the Internet was not designed with security in mind (and it was assumed that every entity was trusted), a substantial body of work has focused on securing the current Internet with patches constrained by the current Internet architecture. However, since incremental improvements have to meet the constraints imposed by the architecture, they often are not optimized with respect to performance, management, etc. Moreover, the architecture itself is the root cause of several security problems. As new applications and concerns are introduced, the original design principles of the Internet may no longer be adequate for Internet users today, and many of the security problems may be difficult to resolve without fundamental architectural changes. For example, the lack of accountability in the current IP layer is to blame for numerous untraceable attacks involving spoofed IP addresses.

Recognizing the potential limitations to patch the current Internet, *clean-slate Internet designs* [178, 2, 11, 72, 176] have received a lot of interest in the research community over the past several years. Instead of patching the current Internet, clean-slate designs are free from the constraints imposed by the current Internet architecture and aim to re-architect the Internet with intrinsic security properties. In particular, our previous work, SCION [178], views security as one major consideration. SCION is a secure and scalable Internet architecture that makes the routing plane more available and resilient against attacks. SCION provides several distinct features—isolation domains, top-down route discovery, and path control—which make it desirable for building a highly available data plane. However, SCION itself does not provide desired availability guarantees.

1.3 Thesis Statement

Following this line of thought, we argue that we can design efficient mechanisms for high availability with a clean-slate approach and make the following thesis statement:

> **On top of a future Internet architecture that supports isolation domains, top-down route discovery, and path control, it is possible to design efficient network-layer mechanisms to mitigate DDoS and address-based selective dropping attacks without keeping per-flow state. In addition, assuming the lower layers are available and a routing path is discovered, these mechanisms can provide waiting time and bandwidth guarantees for flows along the routing path.**

To validate this thesis, we propose defense mechanisms that mitigate DDoS and address-based selective dropping, and show that these mechanisms in concert efficiently provide availability guarantees on top of SCION [178]. Although SCION itself does not provide any availability guarantees for data forwarding, we base our design on SCION because several of its architectural primitives offer substantial advantages for building highly available networks, as Section 2.2 describes. We leave it as future work to formulate these architectural primitives and prove whether these architectural primitives are necessary or sufficient conditions for high availability. We discuss how SCION's primitives help our design in Section 7.3.

1.4 Thesis Overview

Our design is driven by two principles—*isolation* and *fair access to resources*. Isolation protects legitimate traffic from interference by other traffic (including the attack traffic). When isolation is not possible, fair access guarantees that legitimate traffic can obtain a fair share of resources while competing with other traffic.

Following the isolation and fair access principles, we develop complementary technical solutions to four research challenges:

1. One promising approach to isolate DDoS traffic is bandwidth reservation. However, work supporting end-to-end bandwidth reservation in the current Internet faces two fundamental problems: (1) There is no guarantee that a reservation request can reach the destination in the presence of DDoS attacks. In particular, the request can be crowded out by DDoS traffic. (2) Even when the request can arrive at the destination, it is difficult to obtain a meaningful lower bound on the amount of reservable bandwidth for the path. Thus, the research question is: Can SCION's architectural primitives

help address these two fundamental problems? If so, what additional mechanisms are needed to provide meaningful bandwidth guarantees to legitimate traffic despite DDoS attacks?

2. The first part of the work provides a special type of low-capacity channel, on which reservation requests will encounter at most one bottleneck. To bound the waiting time before a successful reservation, the bottleneck AD should handle requests fairly in the sense that the waiting time of the legitimate flows should not be affected by DDoS flows originating from other ADs. More specifically, the research question is: How can we design an efficient mechanism to bound the waiting time to get through a bottleneck on the communication path, thus providing maximum waiting time guarantees?

3. Once bandwidth is allocated and reserved, another important task is to ensure that every flow complies with its allocation. A flow violating its own allocation may compromise the bandwidth guarantees of others. In this part of our work, we ask how to design an efficient monitoring algorithm that helps enforce bandwidth allocation without keeping per-flow state. In particular, we would like to minimize collateral damage due to inaccurate or delayed detection.

4. The waiting time guarantees above are achieved assuming that ADs perform fair dropping of requests in the case of congestion. However, a malicious AD may selectively drop packets (e.g., all dropped packets are from the same sender) regardless of congestion, thus undermining the availability guarantees of the victim. An important research question here is: How can we mitigate address-based selective dropping by malicious ADs?

Scope. The emphasis of this dissertation is on designing novel defense mechanisms to secure packet forwarding (data plane) at the network layer. That is, given one or more discovered routes, this work studies how to move packets from the source to the destination with a high probability of success despite DDoS and address-based selective dropping adversaries. We focus on the network layer for the following reasons. (1) The network layer is often called the narrow waist of the Internet, which is likely to attract attacks because of low protocol diversity. (2) Unlike other layers, the network layer has not changed much since the Internet was first created. (3) The network layer is a critical component that enables global connectivity on the network of networks. We acknowledge that a complete solution of Internet availability requires the protection of all layers against all threats and leave it as future work to study the availability problems in other layers as well as in the control plane.

Although there is an ongoing debate as to whether a clean-slate Internet will ever happen, we believe that the insights and outcomes from clean-slate research can help answer many other research questions.

For instance, clean-slate designs show how much better the Internet could be, which helps researchers and engineers to set a long-term goal and plan the trajectory to get to the goal. Even if a new Internet is never created, our work can be applicable to other availability-sensitive networks, such as smart grids, and the proposed protocols and algorithms can be useful subroutines for many existing applications and systems. For this reason, we try to be as general as possible when describing each of the proposed mechanisms, rather than committing to a certain Internet architecture.

This work can be complemented by detection-oriented mechanisms that identify attackers, as the availability guarantees can be enhanced by blocking, removing, or avoiding the identified attackers. Note that detection-oriented mechanisms alone may be insufficient to provide meaningful guarantees for at least two reasons. First, it is challenging to accurately distinguish benign traffic from malicious traffic, as advanced attacks can mimic the behavior of legitimate traffic. Second, even if every attacker is identified, the victim may be unable to revoke an attacker that is under a different jurisdiction or to avoid an affected area when the network lacks path diversity.

This dissertation concentrates on developing technical solutions and leaves the policy aspects, such as business models and bandwidth distribution policies, to future work. Nevertheless, simple policy examples are provided as needed for ease of explanation.

1.5 Outline

Here we summarize each chapter of this dissertation.

- **Chapter 1** is the introduction. We demonstrate the urgency of solving the problem by showing how vital the Internet is to our economy and society. We identify the major causes of communication disruption, discuss the challenges of building a highly available Internet, and present the thesis statement.

- **Chapter 2** reviews the current Internet architecture and discusses a clean-slate future Internet architecture upon which we build our solutions.

- **Chapter 3** presents STRIDE, a new DDoS-resilient Internet architecture that isolates attack traffic through viable bandwidth allocation, thus preventing a botnet from crowding out legitimate flows. This new architecture presents several novel concepts including tree-based bandwidth allocation and long-term static paths with guaranteed bandwidth. In concert, these mechanisms provide domain-based bandwidth guarantees providing (1) connection establishment with high probability, and (2) precise bandwidth guarantees for established flows, regardless of the size or distribution of the botnet outside the source and the destination domains. Moreover, STRIDE maintains no per-flow state on backbone routers and requires no key establishment across administrative domains.

- **Chapter 4** presents RainCheck Filter, a primitive that bounds the maximum waiting time to get through a bottleneck on the communication path. While RainCheck Filter can be applied to various components at different layers, this chapter considers server flooding as a concrete example. RainCheck Filter is a lightweight DDoS defense primitive that guarantees bounded waiting time for clients despite server flooding without per-client state on the server. RainCheck Filter achieves strong waiting time guarantees by prioritizing clients based on how long the clients have waited—as if the server maintained a queue in which the clients lined up to wait for service. To avoid queuing every incoming client request, the server sends to the client a *raincheck*, a timestamped cryptographic token that not only instructs the client to try again later but also serves as a proof of the client's priority level within a virtual queue.

- **Chapter 5** presents LFD, an anomaly detection algorithm that catches all large flows and avoids false accusation against small flows without keeping per-flow state. That is, this algorithm is *exact outside an ambiguity region*—a configurable region of medium-sized flows. Moreover, most existing algorithms monitor the average throughput over a subset of all possible time windows and thus can be easily bypassed. By contrast, LFD monitors flows over *arbitrary time windows* using leaky-bucket descriptors, thus enabling robust and immediate detection of large flows. These two distinct features yield new applications, such as efficient bandwidth enforcement with minimal collateral damage. Despite its strong properties, LFD is surprisingly scalable because it focuses on accurate classification of large flows and small flows only, and thus can operate at gigabit line rates using a small amount of memory that fits into on-chip SRAM.

- **Chapter 6** presents LAP (Lightweight Anonymity and Privacy), an anonymous forwarding scheme that can be applied to mitigate address-based selective dropping. The core observation is that *making flows indistinguishable from each other prevents the attacker from reacting selectively in the first place*. Popular anonymous communication systems often require sending packets through a sequence of relays on dilated paths for strong anonymity protection. As a result, increased end-to-end latency renders such systems inadequate for the majority of Internet users who seek an intermediate level of anonymity protection while using latency-sensitive applications, such as Web applications. We explore how to achieve near-optimal latency while still achieving an intermediate level of anonymity with a weaker yet practical adversary model such that users can choose between the level of anonymity and usability. LAP is an efficient network-based solution featuring lightweight path establishment and stateless communication. It enhances anonymity against remote attackers by concealing an endhost's topological location.

- **Chapter 7** discusses how to combine the four proposed mechanisms into an integrated solution that provides bandwidth and waiting time guarantees in the presence of DDoS and address-based selective dropping attacks under the assumptions that the layers below the network layer are available and the routing paths are discovered. In a nutshell, STRIDE provides domain-based bandwidth guarantees despite DDoS attacks. Based on the foundation of STRIDE, RainCheck Filter is applied to provide domain-based maximum waiting time guarantees to establish a flow during DDoS attacks. LFD is applied to efficiently catch flows violating their allocation, thus enforcing bandwidth limits without keeping per-flow state. To maintain the guarantees in the face of address-based selective dropping, we adopt LAP to hide network identifiers. Because of the weaker notion of anonymity considered in this work, anonymized packets are still linkable to their respective flows, which makes flow accounting possible.
- **Chapter 8** reviews the related work in DDoS countermeasures, countermeasures to selective dropping, network traffic monitoring, and topology and routing for high availability.
- **Chapter 9** concludes this dissertation. We discuss future directions to further improve Internet availability.

Chapter 2

Background: Internet Architectures

This dissertation aims to improve Internet availability based on a future Internet architecture[1]. Hence, in this chapter, we first review the structure and the interdomain routing protocol of the current Internet and discuss the limited availability guarantees supported by ISPs today. We then review a clean-slate architecture called SCION [178] that we leverage to improve availability.

2.1 The Current Internet

Structure. The Internet is a dynamic collection of interconnected networks called autonomous domains[2] (ADs). Since each AD internally runs a single routing protocol, it can be viewed as a single autonomous entity. We consider an Internet topology at the AD level. In this topology, nodes represent ADs, each of which has several gateway routers (or interfaces) connecting it to neighboring ADs. Links are associated with business relationships, including peering and customer-provider relationships. Two neighboring ADs can have a peering agreement that allows them to exchange traffic for free, or they can have a customer-provider relationship in which the customer AD pays the provider AD to access the rest of the Internet. Endpoint ADs sell Internet access to endhosts.

Interdomain routing. Border Gateway Protocol (BGP) is the de facto interdomain routing protocol (i.e., routing between different ADs) in the current Internet. In general, to move packets from one point to another, a routing architecture implements two primary functions: the control plane and the data plane.

- **Control plane:** The control plane is responsible for discovering and selecting routes. At the AD level, each AD has a routing policy that prioritizes and filters routes going through it. For example, in the

[1] An Internet architecture refers to "any globally agreed upon convention that dictates how packets are handled." [137]
[2] An AD is essentially an Autonomous System (AS) in the current Internet running BGP interdomain routing.

current Internet, an AD may refuse to advertise a provider route to another provider because such transit generates no revenue. An available control plane should derive at least one policy-compliant path between the source and the destination with certain guarantees (e.g., within a bounded time) despite attacks.

- **Data plane:** Given the routes provided by the control plane, the data plane is responsible for the actual packet forwarding. Thus, a forwarding path is a policy-compliant path discovered by the underlying routing protocol. Since forwarding is the main purpose of the data plane, we use data and forwarding planes interchangeably. An available data plane should ensure that the packets arrive at the destination with certain guarantees despite attacks.

Discussion. The current Internet provides few or no end-to-end availability guarantees. Internet Service Providers (ISPs) today vouch for a certain level of availability through Service Level Agreements (SLAs), which state the level of services (e.g., availability, latency, and jitter) that an ISP promises to provide to its customers during the contract period as well as the compensation (e.g., credits for future billing cycles) that will be provided should the ISP fail to achieve the stated level of services. Due to the unpredictable nature of the current Internet, it is common for an SLA to only vouch for the measured performance *over the carrier's own networks* at a coarse time granularity, as demonstrated by this SLA excerpt for AT&T's business class customers[3]: "AT&T Points of Presence (POPs) on the IP/DSL Backbone Network shall be available 99.9% of the time in delivering traffic to/from other AT&T POP locations on the IP/DSL Backbone measured over a calendar month."

2.2 SCION: A Security-Centric Internet Architecture

Among new Internet architecture proposals, we choose SCION as the basis for our design because several of its architectural primitives offer substantial advantages toward building highly available networks. For example, SCION's periodic beaconing, which is used for scalable top-down route discovery, can help propagate timely availability information by which downstream domains can make informed decisions in bandwidth allocation. Packet-carried forwarding state separates routing from forwarding, thus enabling the isolation of data plane failures from control plane failures. Note that despite an elevated default level of availability, SCION is still vulnerable to DDoS and address-based selective dropping attacks; we seek to mitigate these two attacks and to provide availability guarantees by utilizing SCION's architectural primitives.

[3] AT&T Broadband - Business Edition Service Level Agreements. http://www.att.com/gen/general?pid=6622

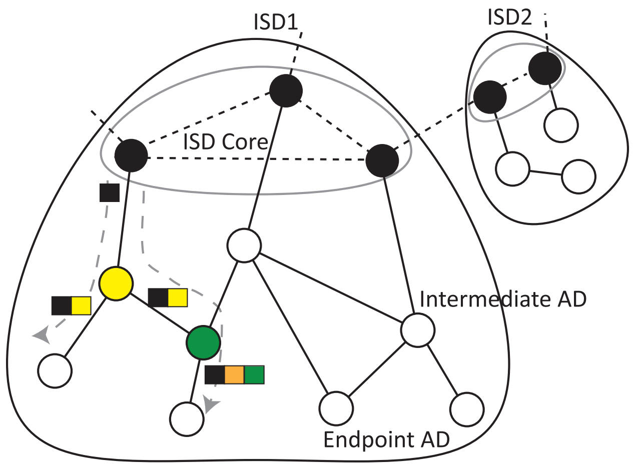


Figure 2.1: Example of an isolation domain (ISD). Each node represents an AD, and the five black nodes represent the tier-1 ADs that constitute the ISD Cores. Each square corresponds to the node's path information.

Structure. Based on the same network structure as that of the current Internet, SCION divides ADs on the Internet into several isolation domains (ISDs), where an ISD is defined as "a set of ADs that agree on a coherent root of trust and have mutual accountability and enforceability for route computation under a common regulatory framework" [178]. Each ISD contains an *ISD Core* consisting of the tier-1 ISPs that manage the ISD. The primary advantage of such a division is that it avoids having a single root of trust for the entire Internet, which is difficult to unanimously agree on in practice. For ease of presentation, we will focus on operations within one ISD unless explicitly mentioned otherwise. Figure 2.1 illustrates these concepts.

Interdomain routing. At a high level, every SCION AD learns a set of half-paths to reach its ISD Core via Path Construction Beacons (PCBs, which we will describe in detail later). The destination publishes some of its half-paths to the Path Server in the ISD Core, such that the source wishing to communicate with the destination can query the Path Server for the destination's path information. ISD Core periodically broadcasts PCBs, which establish half-paths back to the ISD Core as they are disseminated throughout the network in a top-down manner (i.e., from the ISD Core to endpoint ADs). End-to-end communication paths are established by combining the source's and destination's half-paths.

PCBs are constructed as follows: the ISD Core initiates PCBs, which contain *one-hop* paths starting from the core to its adjacent customer ADs with their expiration times. Upon receiving a PCB, an intermediate AD updates the PCB for each of its downstream ADs (e.g., customers and peers) with the authenticated local topology: the i^{th} intermediate AD (AD_i) appends to the PCB the local path information, ingress and egress interfaces I_i, for a particular downstream AD (AD_{i+1}) followed by an opaque field O_i, which encodes the forwarding decision as ingress/egress points at AD_i.

$$\begin{aligned} I_i &= ingress_i \| egress_i \| AD_{i+1}, \\ O_i &= \mathrm{MAC}_{K_i}(I_i \| O_{i-1}), \end{aligned} \tag{2.1}$$

where $ingress_i$ and $egress_i$ stand for the ingress and egress interfaces of AD_i. O_i is computed using a secret key K_i known only to AD_i to protect the integrity of the routing information. Also, AD_i digitally signs PCBs to prevent fake route injection. Note that PCBs in SCION do not announce bandwidth availability.

AD_i propagates the updated PCB to the designated downstream AD (AD_{i+1}). AD_i repeats this process for other downstream ADs on different paths, and upon receiving PCBs, the downstream ADs learn the path to reach the ISD Core. PCBs travel along a special control channel that has isolated bandwidth from all data packets and hence is protected from data-plane DDoS attacks.

For each received PCB, an endpoint AD learns a series of interfaces and opaque fields, which represent the forwarding decisions of the corresponding path. To send packets on the path, the sender embeds in the packet header the forwarding decisions, which remind every intermediate AD of its own routing decision for carrying the packet based on its policy. Hence, no forwarding state at routers is needed.

Among all the half-paths that an endpoint AD learns from PCBs, the endpoint AD selects some as up-paths for reaching the ISD Core and some as down-paths for receiving packets from the ISD Core. To form end-to-end paths, the destination AD publishes its down-paths (i.e., the forwarding states of these down-paths) to the Path Server, which is a DNS-like system, in the ISD Core. A source AD wishing to communicate with the destination can query the Path Server for the destination's down-paths. An end-to-end path is then constructed by splicing the half-paths of the source and the destination.

Although not mentioned explicitly, a SCION ISD Core has to be available all the times for initiating beacons, hosting the Path Server, and forwarding all end-to-end traffic going through the cores. One interesting observation is that by leveraging a small, highly available, trusted network core that is reachable from all other entities in the Internet, we may be able to achieve high availability of Internet-wide communications. As with many security works, our intuition here is to extend guaranteed availability from the core to the entire Internet through cryptographic mechanisms.

Architectural primitives. SCION offers several useful primitives that can facilitate the design of a highly available Internet:

- **Isolation domains.** In computer security, security by isolation is a well-known concept that helps simplify the design of security mechanisms. Applying this concept to network research, SCION segregates mutually distrustful entities into ISDs. This ISD division allows us to explore efficient design for securing intra-ISD communication, which is expected to account for a significant fraction of Internet traffic. Specifically, from a victim's perspective, if the attacker is in the same ISD, the victim can hope for detection and punishment of the attacker as they are under a common jurisdiction. If the attacker is in a different ISD, the victim has to adopt stronger protection that can prevent the attack or make the victim resilient to the attack.

- **Top-down route discovery.** In SCION, the ISD Cores periodically send PCBs, which are signed in an onion fashion for authenticity and non-repudiation. Consequently, PCBs are a secure and reliable vehicle for delivering timely path-based and topological information (e.g., available bandwidth along the path) from upstream to downstream.

- **Path control.** In SCION, both of the source and destination ADs can control the end-to-end path. Path selection and deterministic paths are promising to allow path-based guarantees. Moreover, each packet carries its own forwarding state such that ADs can retrieve the next hop from the packet without keeping local per-flow state. PCFS separates routing from forwarding, allowing us to address their security problems separately. PCFS is extremely useful in many contexts: it can facilitate a variety of applications, such as source-controlled paths and lightweight anonymous forwarding.

Chapter 3

STRIDE

DDoS attacks are a low-cost, high-profile means to interrupt Internet communications. This chapter explores how to provide domain-based bandwidth guarantees to forwarding paths in the presence of DDoS attacks flooding Internet links. Such domain-based bandwidth guarantees protect legitimate traffic from being crowded out by attack traffic.

Motivation: Challenges of mitigating DDoS attacks in the current Internet. The recently proposed Coremelt attack [149] poses a new threat and has not been effectively addressed by any system to date. In an Coremelt attack, an adversary uses a large-scale botnet whose bots communicate only with each other to overload network links. Current DDoS defense mechanisms that attempt to eliminate undesired traffic are rendered ineffective, as all inter-bot traffic is desired by the bot endhosts. Other DDoS defense mechanisms that perform per-source or per-computation fair sharing at congested links may in fact give disproportionate advantage to sources with small uplink bandwidth or with high computational resources, respectively. Moreover, malicious domains can misuse per-source fair sharing by creating multiple bogus senders, and per-computation fair sharing may be too expensive to protect every data packet. Furthermore, global fair sharing implies global fate sharing—a source's share is affected by bots in distant domains over which the source has no influence.

Current DDoS countermeasures have encountered fundamental limitations to address the challenges we describe above to be compatible with the current Internet. Thus, an exciting research challenge is to study if a next-generation network architecture could be more effective against DDoS attacks—what architectural primitives can effectively defend against DDoS attacks?

Our solution: Domain-based bandwidth guarantees on top of SCION. In this chapter, we formulate a new network architecture called STRIDE that provides *domain-based guarantees* for intrinsic DDoS protection within an Isolation Domain (ISD), which contains a set of contiguous Autonomous Domains (AD) with a common root of trust. Specifically, STRIDE provides precise bandwidth guarantees to AD-level paths, or the "sanctuary trails" that isolate attack traffic from legitimate traffic. Each endpoint AD can then internally split the guarantee among its endhosts.

Our architecture is based on the following insights: (1) Bandwidth allocation is simple in a tree-based topology, as the available bandwidth can be split from the root down to each leaf. (2) With network capabilities encoded in packet-carried state and fixed bandwidth classes, routers can perform enforcement in a per-flow stateless fashion using probabilistic detection of the largest flows. (3) By combining a static long-term traffic class guaranteeing low bandwidth with a dynamically-allocated short-term traffic class guaranteeing high bandwidth, we can provide a variety of guarantees to both privately communicating endhosts and public servers.

We leverage the SCION next-generation Internet architecture [178] to perform the tree-based bandwidth allocation: paths are created and available bandwidth is allocated as paths branch out like trees from the network core. The packet-carried forwarding state of SCION also provides us with a natural way to encode network capabilities [12, 168].

Note that prior schemes for bandwidth reservation (e.g., RSVP [23]) or Quality of Service are insufficient to guarantee timely end-to-end data delivery in the presence of DDoS adversaries for several reasons. First, their reservation requests are unprotected against DDoS attacks. Second, they lack lower-bound guarantees for reservable bandwidth. In contrast, STRIDE provides domain-based guarantees, achieving previously unachievable DDoS defense properties for communication within an ISD. Many of the properties also translate for communication between ISDs.

STRIDE provably guarantees connection setup for private communication, sets a probabilistic bound on the waiting time for accessing public services, and provides precise bandwidth guarantees for established flows, all achieved regardless of the size or distribution of the botnet outside the source and destination ADs. These guarantees enable STRIDE to mitigate emerging threats such as the Denial-of-Capability (DoC) and Coremelt attacks. Furthermore, STRIDE does not require backbone routers to keep per-flow state.

3.1 Threat Model

We consider massive Distributed Denial of Service (DDoS) attacks launched by a botnet, which consists of a large number of malware-infected bot endhost machines. In particular, we address two types of DDoS attacks: (1) disabling connection setup in capability-based protocols (DoC attack [15]) and (2) exhausting link bandwidth to crowd out established legitimate connections. In bandwidth exhaustion attacks, we especially focus on the Coremelt attack [149], which aims to overload a target ISP's backbone network using a large number of legitimate-looking flows established among colluding bots (hence, any attempt to identify the attack based on the flow's bandwidth would fail).

3.1.1 Desired Properties

We aim to achieve the following properties for a DDoS-resilient network architecture.

Domain-based guarantees within an ISD. Precise bandwidth guarantees should be provided to communication between endpoint domains residing in the same ISD, and each endpoint domain can internally split the guarantee among its endhosts based on its local policy. A domain that intends to achieve highly available communication could identify malicious bots and remove them to provide better guarantees to legitimate endhosts. Domain-based guarantees ensure that the effect of attacks is confined to infested domains, such that the endhosts can establish a bandwidth-guaranteed flow with high probability.

Robustness and efficiency. To be resilient to DDoS attacks, network elements require efficient protocols and network devices. This indicates that the architecture should avoid per-flow or per-host state at backbone routers and should avoid expensive operations such as digital signature generation or verification in the fastpath.[1]

Flexible route control. Endpoint ADs should be able to control paths. For example, ADs should be able to hide/disclose paths for private/public communication, decide the amount of allocation, and change inbound paths to shift traffic.

3.1.2 Assumptions

In designing a new DDoS resilient Internet architecture, we only make two fundamental assumptions that can be justified using existing security mechanisms.

ISD Cores support congestion-free communication. Since the ISD Core topology is small and relatively fixed, ADs in the ISD Core can accurately assess and provision the capacity requirement of each link to ensure no packet loss for traffic below a certain rate. For example, given a topology and its link capacity, the ISD Core can adopt congestion-free routing [173] to determine how much congestion-free traffic it can support on each incoming link.

[1]Fastpath and slowpath refer to different processing elements that handle packet forwarding in a router. The fastpath refers to packet processing by dedicated hardware, for example, by the linecard in a router. The slowpath refers to packet processing by a general CPU, which often results in a packet processing latency that is usually about two orders of magnitude slower than the fastpath.

ISD Core detects and revokes malicious AD members. Since all ADs of an ISD are within a uniform legal environment (as described in Section 2.2), the ISD Core can revoke the membership of misbehaving ADs. In Section 3.7, we discuss technical approaches to detect compromised or poorly-administered ADs that fail to correctly monitor traffic.

3.2 STRIDE: Design Overview

STRIDE leverages SCION for DDoS resilience. Among new Internet architecture proposals [170, 178], we base our design on SCION because its notion of Isolation Domain (ISD) and secure top-down route discovery enable tree-based bandwidth allocation within a uniform legal environment, whereby bandwidth guarantees and attack isolation can be enforced. Note that although SCION enables natural isolation of attack traffic from untrusted entities, it is still vulnerable to DDoS threats within an ISD. In contrast, STRIDE seeks to provide domain-based bandwidth guarantees for intra-ISD communication.

We first sketch how our new architecture, STRIDE, provides guaranteed end-to-end data delivery to legitimate flows even in the presence of DDoS attacks. In a nutshell, STRIDE protects end-to-end data delivery by establishing communication *channels*[2] that confine the effect of attacks to their originating domains, leaving other domains unaffected. Such communication channels also form the "sanctuary trails" that isolate attack traffic from legitimate traffic. Moreover, since bots within a domain will compete among each other for a fixed amount of bandwidth, bandwidth guarantees can be provided to channels regardless of the size or distribution of the botnet outside the source and destination domains.

Hence, the primary challenges STRIDE faces are how to secure such channel establishment and adjust allocations in response to dynamic traffic patterns. STRIDE addresses these challenges by combining (1) a low bandwidth but long-lived and guaranteed traffic class for channel establishment with (2) a high bandwidth but short-term dynamically-allocated traffic class. Specifically, a channel is constructed using any of the three types of *bandwidth classes*: static, dynamic, and best effort (BE). Hybrid channels are also possible, as we explain later. For simplicity, we focus on presenting STRIDE using bidirectional links and elaborate how STRIDE can support directional links in Section 3.7.

Bandwidth classes. As shown in Figure 3.1, link bandwidth is split up into three bandwidth classes:

- *Static class* is for guaranteed, persistent long-term bandwidth that ADs allocate, for example to protect initial connection setup request packets between a source and a destination. Each AD allocates a small portion of its total bandwidth (e.g., 5–15%) to this class.

[2]The bandwidth of a *path* is divided into separate *channels*, and multiple channels dynamically share the bandwidth within a path.

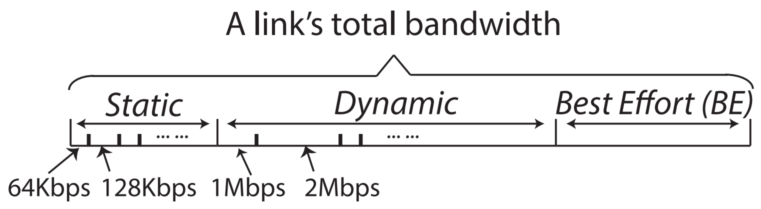


Figure 3.1: Three bandwidth classes of STRIDE.

- *Dynamic class* is for guaranteed, short-term end-to-end bandwidth allocations and supports high-capacity channels. The dynamic class may account for the majority of the link capacity (e.g., 60–65%).

- *Best-effort (BE) class* is allocated with the remainder of the bandwidth (e.g., 30%). It can take over the unused bandwidth from the other (uncongested) classes using statistical multiplexing.

Within the static or dynamic bandwidth class, an AD can assign different bandwidth subclasses (e.g., 500 Kbps, 1 Mbps, etc.) to individual paths/flows based on an empirically measured flow size distribution. For example, the fraction of dynamic bandwidth allocated to the 1 Mbps subclass can be derived based on the fraction of flows with 1 Mbps rate in the current Internet. We provide guidelines on how an AD can divide its total link capacity to the above three classes in Section 3.7.

A half-path announced by a PCB is a path on the BE class, thus offering no guarantees. However, for each provider AD, an endpoint AD can *activate* up to k BE half-paths to convert each into a static half-path, which offers a guaranteed amount of bandwidth from that AD to the ISD Core. The parameter k is determined by contract and is enforced by provider ADs.

Static and BE channels. A communication channel is a conduit to carry traffic from a source to a destination. A single half-path can be used as a channel to reach the ISD Core, and two half-paths can be combined to form an end-to-end channel between a source and a destination. In this chapter, we call the half-path of the source the up-path (as traffic on that channel traverses ADs upwards towards the ISD Core) and the half-path of the destination the down-path. If two BE half-paths (i.e., half-paths using the BE bandwidth class) are combined, the resulting channel is a BE channel, and similarly, two static half-paths create a static channel. Hybrid channels are also possible, and we will make use of a static up-path that is combined with a BE down-path. Packets flowing through different types of channels have different properties.

Dynamic channel. An endhost can send a request on static, BE, or hybrid channels to reserve a dynamic channel, which provides high bandwidth for a short amount of time (on the order of seconds to

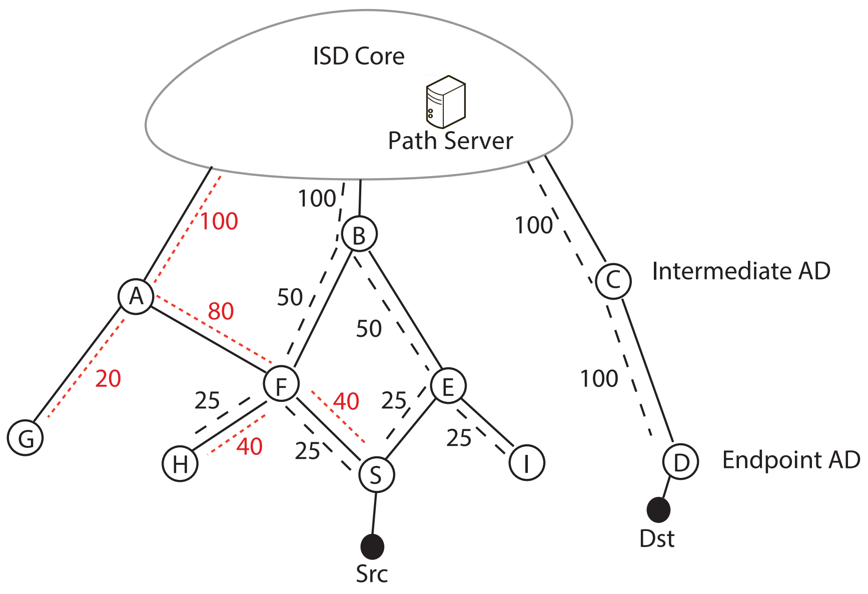


Figure 3.2: Illustration of STRIDE and how bandwidth is split and announced through PCBs.

enable fast revocation). All ADs, including the destination, need to agree on the amount of bandwidth offered for the dynamic channel. This channel is similar to network capabilities [12, 168] with bandwidth guarantees [171].

One surprising aspect of STRIDE is that only the access routers in the endpoint ADs have to keep per-flow state for flow admission and policing. STRIDE does not require intermediate routers to maintain any per-flow or per-path state in the fastpath for forwarded traffic; only the initial channel establishment requests require slowpath operations for admission control. Performing per-flow management at edge routers is shown to be practical [148], and we can further relax this state requirement using probabilistic detection of the largest flows. With these ingredients, we construct a series of mechanisms that achieve the highly available communication we strive for.

3.2.1 Static Half-Path Setup

We now describe how an endpoint AD (AD_S) establishes a guaranteed static path (i.e., a path using static bandwidth class) to its ISD Core.

① **Bandwidth announcement:** The ISD Core assesses its link capacity and adds information regarding current reservable static bandwidth to periodic PCBs (ISD Core and PCBs are explained in Section 2.2). The ISD Core ensures no congestion on its internal links even if the announced static bandwidth becomes fully reserved. As a PCB travels from the ISD Core to endpoint ADs, each AD adds information regarding its own bandwidth availability. In particular, an intermediate AD splits the reservable bandwidth and announces the split amount for each of its children for the static path. Figure 3.2 depicts the bandwidth availability announcement (denoted by the numbers) of PCBs for the static path. This particular diagram shows that AD_S has three possible static paths:

❶ $ISDCore \overset{100}{\rightarrow} AD_A \overset{80}{\rightarrow} AD_F \overset{40}{\rightarrow} AD_S$

❷ $ISDCore \overset{100}{\rightarrow} AD_B \overset{50}{\rightarrow} AD_F \overset{25}{\rightarrow} AD_S$

❸ $ISDCore \overset{100}{\rightarrow} AD_B \overset{50}{\rightarrow} AD_E \overset{25}{\rightarrow} AD_S$

② **Activation:** Recall that each endpoint AD learns a set of BE half-paths to the ISD Core through PCB propagation. The AD can then activate k BE half-paths per provider AD to static class as follows. Upon receiving a PCB with the bandwidth availability information, AD_S sends an activation request to the ISD Core to reserve the static bandwidth as specified in the PCBs. While forwarding this request, all the intermediate ADs temporarily reserve the requested static bandwidth for AD_S if they can support the request. For example, AD_S may request 40 units of the static-class bandwidth through path ❶, 25 units through path ❷, etc.

③ **Confirmation:** The ISD Core sends a confirmation to AD_S for the guaranteed static path that AD_S (and all of its hosts) can use to reach the ISD Core. The new opaque fields constructed along the confirmation enable AD_S to communicate with the ISD Core on the static path. Note that STRIDE allows each endpoint AD to create up to k static half-paths per provider AD, and each endpoint AD has the freedom to keep a subset of static half-paths in private for privileged access, as we will explain in step ④ below, while registering others at the Path Server for public usage.

3.2.2 Static and BE Channel Setup

After the half-path setup, each endpoint AD obtains a set of static bandwidth-guaranteed half-paths in addition to a set of BE half-paths provided by the original PCBs. We now describe how two half-paths (i.e., an up-path and a down-path) can be combined to setup an end-to-end channel. Combining static and BE half-paths results in four types of channels (i.e., static, BE, static+BE, or BE+static) with different guarantees, as summarized in Table 3.1.

④ **End-to-end path selection:** When source *src* of AD_S wants to communicate with destination *dst* of AD_D, *src* queries the Path Server for the down-paths to reach *dst*. *Src* reaches the Path Server in the ISD Core using a BE or static half-path as a communication channel. The Path Server then returns unconcealed static down-paths and/or BE down-paths to *src*. Alternatively, *dst* can inform *src* of a private static path over an Out-Of-Band (OOB) channel. By combining one of its up-paths and one of the down-paths provided by the Path Server, *src* now establishes an end-to-end channel for sending a dynamic channel setup request.

3.2.3 Dynamic Channel Setup

⑤ **Dynamic channel setup request:** To acquire guaranteed bandwidth along a dynamic channel, *src* sends a dynamic channel setup request on this newly established end-to-end (static, BE, or hybrid) channel. While this request travels toward *dst*, all ADs on the path specify the bandwidth that they can provide for the dynamic channel and forward it toward *dst*.

In the situation that *src* sends packets beyond the allocated static bandwidth of the static up-path, AD_S sets an overuse bit on each extra packet to utilize unused static or BE bandwidth, thereby indicating that the extra packets are beyond the permitted allocation for efficient traffic policing.

If *src* cannot send the request on the static channel, possibly due to congestion on any of the announced down-paths, STRIDE flexibly allows the endpoint AD to send requests on the BE channel. We discuss several alternatives for channel composition and their priorities in Section 3.3.3.

⑥ **Dynamic-class bandwidth allocation:** When *dst* receives *src*'s dynamic channel setup request, *dst* can deduce sustainable dynamic-class bandwidth for *src* based on the reported dynamic-class bandwidth availability of all the intermediate ADs (e.g., minimum of the dynamic-class bandwidth allocations of all the intermediate ADs). Then *dst* sends this information to *src*, during which the *dynamic capability* is constructed on the return path. STRIDE provides flexible options for sending the reply. For example, *dst* may send the reply through the allocated dynamic channel or through the reverse channel that sent the request.

⑦ **Guaranteed data transmission:** When *src* receives the reply, it can enjoy sending data traffic using the dedicated dynamic-class bandwidth by embedding the dynamic capability in the data packets. Since this bandwidth is short-lived, *src* may renew this dynamic-class bandwidth using actual data packets. Similar to step ⑤, *src* can also send more than the permitted dynamic bandwidth allocation, in which case AD_S sets the overuse bit for the extra data traffic.

3.3 STRIDE Protocol Description

In this section, we elaborate on STRIDE's mechanisms.

3.3.1 Static Half-Path Setup

① **Bandwidth announcement:** Upon receiving a PCB, an intermediate AD (AD_i) forwards it to the downstream AD (AD_{i+1}) after appending the bandwidth information, which includes (1) currently reservable static-class bandwidth for AD_{i+1}, (2) currently underutilized static-class bandwidth that AD_{i+1} can use, (3) currently available dynamic-class bandwidth, and (4) currently available BE-class bandwidth. This bandwidth information enables downstream ADs and endhosts to deduce congestion status and make informed decisions in selecting paths. When a PCB reaches an endpoint AD, it contains a path from the ISD Core to the endpoint AD with reservable bandwidth that can be provided once this path is activated (details in step ②).

On each PCB, AD_i adds an opaque field O_i, as described in Eq. (2.1). The resulting collection of opaque fields in the PCB access the BE channel on the route that the PCB traversed. Note that AD_i should use different MAC keys to construct opaque fields for different bandwidth classes and expiration time, so that an attacker cannot forge an opaque field for another traffic class or extend the expiration time. For example, the MAC key K_i can be derived from a master secret key $\hat{K}_i$: $K_i = F_{\hat{K}_i}(BE, timestamp)$, where $F(\cdot)$ is a pseudo-random function.

Path diversity vs. quality. The bandwidth announcement mechanism in STRIDE enables an AD to divide bandwidth among its customers. However, bandwidth allocation is still challenging because of the tradeoff between path diversity (i.e., the number of different paths) and path quality (i.e., the bandwidth allocated to each path). An intermediate AD can offer higher path diversity by propagating a PCB to more children (i.e., egress routers). Yet, increasing path diversity reduces the bandwidth that can be allocated to each child AD since the bandwidth contained in a PCB must be split for all children *recursively* as the PCB propagates to downstream ADs.

To address this issue, STRIDE uses a *bandwidth overbooking* technique to enhance path diversity and quality simultaneously. The observation is that since an endpoint AD can choose up to k paths out of all announced paths, not all announced paths will be activated. Hence, intermediate ADs can announce greater reservable bandwidth (i.e., overbook) to their downstream ADs than the actual link capacity and defer the actual bandwidth reservation later during the activation step (②). However, if intermediate ADs overbook their bandwidth aggressively, path activation could be frequently denied. To address this issue, STRIDE allows each intermediate AD to overbook its bandwidth such that the probability of path-activation failure along its link is below a certain threshold, which we analyze in Section 3.7.

From the bandwidth announcement in the latest PCB, an endpoint AD learns the amount of static bandwidth it may reserve on the corresponding route. Note that the actual allocation is not performed until the activation step (②), and the advertised bandwidth may be greater than what an intermediate AD can support because of overbooking or stale bandwidth information.

② **Activation:** An endpoint AD requests for a path activation along the reverse path to the ISD Core. Each request consists of the desired (1) expiration time of the path and (2) amount of static-class bandwidth. The desired bandwidth should not exceed the announced reservable bandwidth in the latest PCB. STRIDE considers an activation request as a control message, like PCBs, that is protected from data-plane DDoS attacks. To avoid congestion on the control plane, each AD can rate-limit the activation requests on a per customer basis and advertise the limit with the reservable bandwidth during the announcement.

Upon receiving an activation request, an intermediate AD with sufficient unallocated bandwidth (i.e., spare capacity $\geq$ desired bandwidth) temporarily allocates the requested bandwidth for this path. Otherwise, the AD sends back an error message. Also, to minimize bandwidth waste, the AD recycles temporarily allocated bandwidth when the activation fails, which is indicated by the lack of a confirmation (step ③) or an error message before the arrival of the next PCB.

For efficient bandwidth management (e.g., allocation and recycling), the expiration time and bandwidth is chosen from a pre-defined finite set of values. For example, the expiration time can be 6, 12, 18, or 24 hours; the subclass bandwidth assigned to each activation request can be 64 Kbps, 128 Kbps, etc.

Each endpoint AD is allowed to activate up to k distinct paths per upstream AD (or provider). This policy is made in accordance with the observation that in the current Internet, large endpoint ADs often subscribe to multiple providers for increased capacity and path diversity. This k-path policy can be enforced either by the providers or the Path Server in the ISD Core.

③ **Confirmation:** The ISD Core informs the endpoint AD of a successful static path activation by sending a confirmation message along the activated path. The confirmation message contains the expiration time and the allocated bandwidth. The ISD Core also updates the Path Server to include this activated path. Upon receiving the valid confirmation from the ISD Core, each intermediate AD on the path converts the temporarily-allocated bandwidth to be long term (until the expiration time).

Before forwarding the confirmation to the next hop, the AD adds a new opaque field using a different MAC key K_i, derived from the master secret key $\hat{K}_i$ similarly as before: $K_i = F_{\hat{K}_i}(static, timestamp, BW)$, where BW is the amount of bandwidth allocated to the path. After receiving the confirmation, the endpoint AD can forward packets on the static channel of the path by including the new opaque fields in the header of packets.

3.3.2 Static and BE Channel Setup

After the half-path setup, endpoint ADs learn multiple long-term, bandwidth-guaranteed static half-paths (in addition to multiple BE half-paths) to communicate with the ISD Core.

④ **End-to-end path selection:** When an endhost *src* in AD_S attempts to make a connection to another endhost *dst* in AD_D, *src* contacts AD_S for path resolution. In turn, AD_S requests the Path Server in the ISD Core for a list of static paths to *dst*, and the server returns down-paths. As a result, the AD_S can select an up-path (from itself to its ISD Core) and a down-path (from the ISD Core to AD_D), and splice them to form an end-to-end path.

Path Server availability. A successful DDoS attack against Path Servers would disable end-to-end path establishment in STRIDE. Such an attack can be mitigated in several ways: (1) the ISD Core can detect the origin of attack traffic against Path Servers and throttle their traffic, (2) the bandwidth-guaranteed static paths can be used to contact a Path Server, and (3) the Path Server functionality can be distributed to several machines.

Path selection policy. If AD_S keeps on selecting paths based on the highest available bandwidth, it may end up selecting a single best path for all the source endhosts (besides *src*), eventually congesting this path. To resolve this issue, endpoint ADs in STRIDE perform probabilistic path selection as follows: the endpoint AD selects a path with a probability proportional to the path bandwidth guarantees. With this policy, the endpoint ADs are more likely to select uncongested paths and reduce the average number of trials. We evaluate a specific instance of this policy in Section 3.5.

Private paths. We introduce the notion of private paths, which endpoint ADs can use to provide guaranteed down-paths to preferred endhosts. In a nutshell, an endpoint AD can keep some half-paths as private and provides them to its endhosts such that they can selectively provide them to preferred sources. We define private services to be provided by those servers that can predict future customers (e.g., premium customers on Amazon). A private server providing access to a closed community can provide guaranteed connection setup to community members with private down-paths as follows: a destination can selectively disclose its private down-paths to preferred sources. The private paths can be distributed via OOB channels or by uploading encrypted private paths to a Path Server. As a result, a valued customer of Amazon, for example, can obtain a bandwidth-guaranteed static down-paths for sending dynamic channel setup requests to Amazon.

Table 3.1: Guarantees of dynamic channel setup delay and bandwidth for different types of end-to-end channels.

Up-path	Down-path	Delay	Bandwidth guaranteed?
Static	Static (private)	Constant	✓
Static	Static (public)	Linear	✓
Static	Best-effort	Linear	✗
Best-effort	any	✗	✗

3.3.3 Dynamic Channel Setup

Using a (BE, static, or hybrid) channel, *src* sends dynamic channel setup requests to establish an end-to-end dynamic channel. With such an end-to-end dynamic channel, STRIDE can provide bandwidth guarantees to short-term, high-bandwidth dynamic flows. Note that *src* can send any types of packets on the end-to-end channel, but we focus on the discussion of sending dynamic channel setup requests, as it is a part of our DDoS defense mechanism.

⑤ **Dynamic-channel setup request:** After selecting an end-to-end channel in step ④, a source endhost can send a dynamic channel setup request for guaranteed dynamic-bandwidth allocation. Table 3.1 describes guarantees of dynamic channel setup delay and bandwidth for different types of end-to-end channels.

A request header carries two additional indicators that enable congested intermediate ADs to efficiently control link bandwidth:

- *Overuse bit:* A source AD sets an overuse bit of a packet on a static up-path in case its endhost is sending packets more than the reserved static-class bandwidth of the up-path.
- *Congestion bit:* Any AD that experiences link congestion sets a congestion bit in BE packets.

Traffic priority of requests. When an AD receives more packets than what its outgoing links can afford, the AD has to discard some of them while maintaining the static-class bandwidth guarantee. Based on where the congestion occurs, we discuss different techniques to prioritize packets.

- *Host contention at source ADs:* Static bandwidth contention may occur on the source AD's outgoing links. Each AD can have a different way to resolve contention. For example, it can adopt a payment-based scheme in which each client informs its host AD how much it is willing to pay for using a static up-path, and the endpoint AD can arrange based on some objectives (e.g., maximize the AD's revenue) [162]. For ease of analysis, we consider per-host fair sharing within an endpoint AD.
- *Link congestion on up-paths:* During link congestion, each source domain obtains *a weighted fair share* of the available (unallocated, or allocated but unused) static bandwidth. A weighted fair share is

Table 3.2: Traffic priority of dynamic channel setup requests on the down-paths that experience link congestion.

Priority	Up-path	Bits set	How requests arrived
1	Static	-	Within allocated static BW
2	BE	-	On uncongested BE link
3	Static	Overuse	Beyond allocated static BW
4	BE	Congest	On congested BE link
5	Outside ISD	-	From outside ISD

proportional to the source domain's static allocation on the congested link. Hence, static packets with the overuse bit would be transmitted on the static channel up to the weighted fair share, and packets beyond the share are converted into BE packets to compete with the standard BE traffic.

- *Link congestion on down-paths:* STRIDE assigns priority levels to the packets such that low priority packets are dropped first in the case of congestion. Table 3.2 summarizes the priority levels ordered from the highest to the lowest. This priority applies to both static and BE classes. If the congestion persists within the first-priority traffic, the congested link assigns to it a weighted fair share of the available static-class bandwidth proportional to each *destination domain*'s static allocation.

Determine reservable dynamic bandwidth. Recall that the dynamic bandwidth class has defined bandwidth subclasses, such as 512 Kbps, 1 Mbps, 2 Mbps, etc., and each dynamic channel is associated with a given subclass. Intuitively, to provide precise bandwidth guarantees for established flows, we have to ensure that every STRIDE-protected request packet can be offered sufficient dynamic bandwidth (e.g., at least 512 Kbps). Ideally, we would like to provide guaranteed flow bandwidth to every request packet traversing static channels, as Table 3.1 shows.

One key challenge here is how to flexibly determine the amount of reservable dynamic bandwidth for such requests. To address this challenge, STRIDE limits the rate of the dynamic channel setup requests (thus the dynamic allocation) within the static class to be *proportional to the static allocation*. For example, if each dynamic channel is guaranteed 10 units per second and expires in 2 seconds, and the rate limit is 3 requests per second, then the dynamic-class link bandwidth should be greater than $10 \cdot 2 \cdot 3$ units per second to accommodate the worst case where all requests arrive using the static class.

The AD assigns the smallest subclass to the initial request and flexibly upgrades to a higher subclass for the subsequent requests for the allocation renewal if the link is not congested. In the case of link congestion on the up-path (down-path), each source (destination) domain obtains a weighted fair share of the available dynamic-class bandwidth that is proportional to the source's (destination's) static allocation on the congested link.

Each AD on the path (including the destination AD) either approves the requested dynamic bandwidth or reports the maximum available bandwidth (which is at most the available bandwidth indicated by the previous AD).

⑥ **Dynamic-class bandwidth allocation:** Through a dynamic-channel setup request, a destination end-host can discover the bottleneck link(s) and the available dynamic-class bandwidth along the path. The destination constructs a reply packet that carries (1) reserved dynamic-class bandwidth of this flow, (2) opaque fields, and (3) expiration time which indicates the lifetime of the guaranteed dynamic bandwidth. The destination also indicates which AD-to-AD link(s) is the bottleneck for determining the bandwidth reservation.

As the packet travels back to the source, ADs update their dynamic bandwidth allocation and opaque fields to accurately reflect the available bandwidth and reduce the potential waste of bandwidth. If the allocated end-to-end dynamic bandwidth does not meet the source's need (e.g., determined by an application service), the source may select an alternative path. Furthermore, the source can make an informed decision to avoid the bottleneck link when selecting an alternative path.

Dynamic channel update. A source can renew the short-term dynamic channel while communicating with the destination. The sender sets a renewal bit in the header of the dynamic-class packets. If the destination renews, the source AD invalidates the old channel (e.g., by keeping track of the latest channel for each flow and rejecting packets using old channels) to prevent misuse.

⑦ **Guaranteed data transmission:** Upon receiving the reply, *src* can use the end-to-end dynamic channel for guaranteed data transmission. *Src* can also flexibly choose other types of end-to-end channels for different guarantees.

Regulation. For per-flow bandwidth guarantees, endpoint ADs monitor per-flow data usage and regulate potential violation. For example, every endpoint AD ensures that the overuse bit is set in every data packet whose flow rate exceeds the allocated value. ADs are responsible to drop some of the data packets with the overuse bit to resolve link congestion. For example, similar to the static channel regulation, the AD can drop packets that are beyond the weighted fair share of the source or the destination. In addition, intermediate ADs and the ISD Core can perform both real-time probabilistic monitoring and offline traffic analysis to identify misbehaving endpoint ADs that fail to regulate their clients. ISD Cores and ADs also monitor per-ISD bandwidth usage of dynamic-class traffic at each interface at the ISD boundary to isolate attack traffic from other ISDs.

3.4 Bandwidth Guarantee Analysis

We first show that STRIDE achieves domain-based guarantees for communication between the source (AD_{src}) and the destination (AD_{dst}) domains within an ISD. Specifically, we analyze what domain-based guarantees AD_{src} can obtain using different types of channels. We then discuss how an endpoint AD can divide such domain-based guarantees among its endhosts.

In Theorem 1, we show that by leveraging private down-paths, AD_{src} and AD_{dst} can establish bandwidth-guaranteed static channels for congestion-free communication. Let u_i and d_i be AD_i's total static up-path and down-path bandwidth allocations, respectively, where $1 \leq i \leq m$ and m is the number of ADs. Since each AD is expected to assess its bandwidth requirement of static half-paths based on its contractual agreements with human subscribers, u_i and d_i are constant irrespective of the number of ADs or the power of the botnet. We denote $d^p(i, j)$ to be the total bandwidth of AD_j's private down-paths known only to AD_i.

Theorem 1. *For private communication (using private static down-paths), AD_{src} can successfully send packets to AD_{dst} at rate $r^p = \min\{u_{src}, d^p(src, dst)\}$ without experiencing congestion on any intermediate links.*

Proof sketch: The first domain-based guarantee is straightforward. Since ISD Core is congestion-free (as described in Section 3.1.2), AD_{src} can establish end-to-end congestion-free channels by splicing its static up-paths and AD_{dst}'s private static down-paths. The sending rate of the resulting channels is dominated by the bottleneck bandwidth, which is the minimum of u_{src} and $d^p(src, dst)$. Both u_{src} and $d^p(src, dst)$ are independent of the botnet and other ADs' allocations. $\square$

Note that r^p is a lower-bound guarantee of the sending rate, and the congestion-free property ensures that packets, such as connection setup requests, can be delivered at the first trial.

In Theorem 2, we show that AD_{src} can obtain a weaker guarantee (which depends on the static allocations of other ADs) when using public static down-paths. Let $U = \sum_{i=1}^{m} u_i$, and $U(i)$ be the total static up-path bandwidth activated by ADs that desire to communicate with AD_i.

Theorem 2. *For public communication (using unconcealed static down-paths), AD_{src} can successfully send packets to AD_{dst} at an average rate $r = u_{src}\frac{d_{dst}}{U(dst)}$.*

Proof sketch: Sources that desire to communicate with AD_{dst} compete for the limited bandwidth of static down-paths, d_{dst}. Sources do not need to compete with packets to other destinations on congested links because STRIDE performs weighted fair sharing on the static down-paths. To obtain the highest traffic priority and increase the chance of successful delivery, each source sends packets with no overuse bit on its static up-paths at full speed, resulting in $U(dst)$-amount of high-priority traffic to AD_{dst}. Hence, AD_{src} with u_{src} static allocation can send packets through static down-paths at rate $\frac{u_{src} \cdot d_{dst}}{U(dst)}$ (e.g., bit/s) on average.

$\square$

Theorem 3 shows that AD_{src} can obtain a lower-bound guarantee on the dynamic allocations to AD_{dst}. Let γ be the ratio of the dynamic-class bandwidth to the static-class bandwidth, and we assume γ is the same for every link for ease of description. $\Delta(i, j)$ is the guaranteed dynamic-class bandwidth that AD_i would like to allocate for communication with AD_j.

Theorem 3. *For dynamic channels, STRIDE guarantees* $\min\{\Delta(src, dst), \Delta(dst, src)\}$ *amount of dynamic-class bandwidth for the flow aggregate between* AD_{src} *and* AD_{dst}*, where* $\sum_{i=1}^{m} \Delta(src, i) \leq \gamma \cdot u_{src}$ *and* $\sum_{i=1}^{m} \Delta(dst, i) \leq \gamma \cdot d_{dst}$.

Proof sketch: Because STRIDE performs weighted fair sharing within the dynamic class based on the static allocation, the flow aggregates from AD_{src} and to AD_{dst} are guaranteed to have $\gamma \cdot u_{src}$ and $\gamma \cdot d_{dst}$ bandwidth, respectively. Endpoint ADs can then freely divide the guaranteed bandwidth to flow aggregates going to/from different ADs. $\square$

Splitting guaranteed bandwidth. Each endpoint AD decides how to divide its bandwidth guarantees among its endhosts based on its local policy. For example, a simple policy would be to split the static allocation based on per-host fair sharing. Suppose the sender and the receiver obtain a fair share u'_{src} and d'_{dst}, respectively, from their local domains. Similar to Theorems 1–3, we can show guarantees for the sender and receiver by replacing u_{src} with u'_{src} and d_{dst} with d'_{dst} in the proofs above. Moreover, because the number of dynamic channel setup requests within the static class is small and bounded, STRIDE can provide guaranteed high bandwidth to flows established through static channels. An interesting observation is that since compromised endhosts send traffic all the time whereas uncompromised endhosts do not, the available bandwidth is much higher than the fair share if the domain contains fewer bots.

Resilience against DoC, Coremelt, and Crossfire Attacks. A Denial of Capability (DoC) attack aims to disrupt the delivery of capability requests, or the dynamic channel setup requests. Theorems 1 and 2 imply that a DoC attacker (1) cannot crowd out a capability request if the request is placed along a static channel with a private down-path and (2) can delay a capability request at most by time linear to the static allocation of other domains if the request uses a public down-path. These bounds are independent of the size or distribution of the botnet outside the communicating ADs.

In the Coremelt attack, bots send traffic among each other, overloading a backbone link by a quadratic number of low-intensity flows that are wanted by the destinations. Theorem 3 implies that STRIDE can defend against Coremelt attacks because the guaranteed flow bandwidth between the source and destination ADs is unaffected by bots outside those ADs.

In particular, STRIDE linearizes the Coremelt attack through domain-based isolation. Without STRIDE, a legitimate flow has to compete with attack traffic that grows quadratically with the number of bots. With STRIDE, a legitimate flow only needs to compete with attack traffic that grows linearly with the number of bots in the source AD, assuming the source AD limits the rate on a per-sender basis rather than per-flow. That is, the legitimate flow is isolated from attack traffic between bots outside the source and destination ADs. Moreover, provided that users can opt to subscribe to a clean AD, the market mechanism incentivizes ADs to clean up their own domains.

The Crossfire attack cuts off a target area by flooding a few selected links. This attack exploits a characteristic of the current Internet—a few critical links are responsible for a large portion of traffic to a target area. The Crossfire attack exhibits several distinct properties. First, bots send traffic to decoys (e.g., public IP addresses) rather than other bots, increasing the pool of attack flows. Second, the attack floods links across different ISPs, such that it cannot be mitigated by individual ISPs. Third, to attack persistently, it is designed to avoid any route change.

It is still an open research question whether the Crossfire attack can succeed on new Internet architectures that support multiple route choices and route control, such as SCION. Such new architectures may be able to weaken Crossfire attacks by either distributing traffic among a large set of routes or quickly switching away from congested routes. Hence, we examine whether STRIDE can counter Crossfire assuming the attack can succeed on SCION.

If the overloaded links are not in the ISD Cores, STRIDE can similarly isolate attack traffic from bots outside the source AD or to decoys outside the destination AD and provide guarantees depending on the bots in the source AD only. The problem is thus reduced to whether the attack can overload the ISD Cores. While this work assumes the ISD Cores can accurately provision and control the advertise bandwidth so as to be free from congestion, we leave it as future work to design concrete mechanisms for congestion-free routing within and between ISD Cores.

3.5 Evaluation

In this section, we evaluate STRIDE with respect to its effectiveness against DDoS attacks. We show the effectiveness of end-to-end bandwidth guarantees under large-scale attack scenarios. We also test the packet forwarding performance of STRIDE via real-field implementation.

Simulation setup. For realistic simulation, we use a CAIDA AS-relationship dataset to construct an ISD. A tier-1 AD connecting to 2164 endpoint ADs is chosen as the ISD Core. Although the AS-relationship dataset does not include all interface-level paths, our analysis of the dataset reveals that AD-level path diversity is high enough to support STRIDE's path control and hence to evaluate STRIDE's path construction. Specifically, the endpoint ADs in the dataset have more than 40 different paths to the ISD Core on average. If interface-level paths are constructed, path diversity at endpoint ADs would become much higher since the number of paths grows exponentially as PCBs propagate downstream.

Bandwidth allocation. During PCB propagation, each AD allocates bandwidth to each child AD proportional to the child size. We assume that the size of an AD is proportional to its degree.

3.5.1 Resilience against DoC Attacks

We evaluate the resilience of STRIDE against DoC attacks under the following simulation scenario. We randomly label one hundred ADs as *clean* (i.e., the ADs containing no bots) and configure them to send traffic to a destination AD using 10 different down-paths (i.e., k=10), with a send rate equal to one tenth of the down-path capacity. Hence, in the absence of attacks, all down-paths are fully (but not overly) utilized. Then, we randomly label ADs as *contaminated* (i.e., the ADs containing bots) and set their send rates equal to that of a clean AD so as to make individual contaminated ADs indistinguishable from clean ones. The number of contaminated ADs is increased from 0 to 300.

We evaluate the effectiveness of STRIDE against the above attacks in the following two scenarios.

1. *Public Paths:* All clean and contaminated ADs use their k activated down-paths to setup capabilities with the destination AD.

2. *Public and Private Paths:* Half of the clean ADs use a private down-path that was provided to the source ADs via a secret out-of-band channel. Meanwhile, the remaining half of the clean ADs and contaminated ADs use the public paths as before.

We use the *admission ratio* as an evaluation metric. Admission ratio is defined as the percentage of the legitimate packets (i.e., packets from the clean domains) that successfully traverse the bottleneck link/path.

Figure 3.3 shows that when all source ADs use the public paths ("All"), the admission ratio of the legitimate packets decreases as more contaminated ADs are added since the per-AD bandwidth decreases. When half of the clean ADs acquire a private path from the destination ("Private"), their packets are

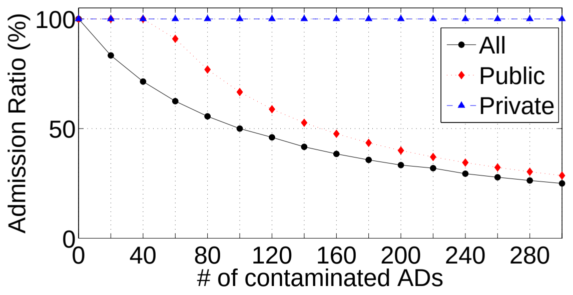


Figure 3.3: The admission ratio of legitimate packets for the number of contaminated ADs.

unaffected by the attack as the 100% admission ratio shows; the packets of the remaining half of the clean ADs ("Public") obtained higher admission ratio along the public paths because the use of the private path reduced bandwidth contention along the public paths. This result illustrates how destination ADs can protect their valued customers' traffic from DDoS attacks in STRIDE.

While STRIDE enables private parties to use private paths to avoid congested static paths, it also protects clean ADs' traffic from large-scale DDoS attacks via packet prioritization. That is, capability requests made through the static up-paths would have a higher priority than others through the best-effort up-paths. To examine this, we use the following simulation scenario. The attack strength is increased (by adding more attack sources within contaminated ADs) up to 10 times the bandwidth of the static up-paths. Source ADs put the high priority marking on their outbound packets such that the bandwidth of high priority packets would not exceed that of the static up-paths (e.g., if the attack strength grows 10 times, 90% of attack packets would have a low priority marking). Legitimate source endhosts, on identifying congestion on static down-paths, use the best-effort down-paths, and attack source endhosts use the same path selection strategy as that of the legitimate sources to maximize their effects. The above attack scenario is the strongest attack scenario we consider for the given number of legitimate and attack sources since all packets from the 100 clean and 300 contaminated ADs compete for the bandwidth of public paths.

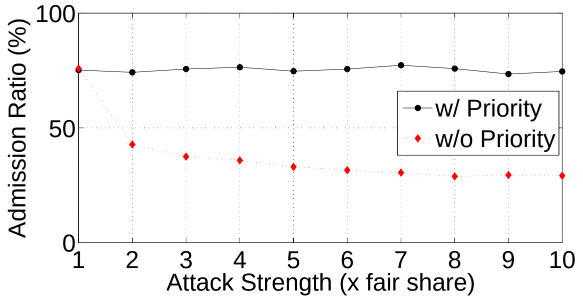


Figure 3.4: The impact of attack strength.

Figure 3.4 shows that even if the attack strength grows, the effects on legitimate traffic are marginal: attack sources, regardless of their strength, can only consume bandwidth proportional to their fair share both on the static and the best-effort channels. Meanwhile, 25% of legitimate packets sent through the static down-paths reach their destination without loss, and the other legitimate packets (i.e., 75% of them) sent through the best-effort channel reach the destination with a ratio close to 66.7%. Overall, 75% of the legitimate requests overcome the massive DDoS attack whose total send rate is 30 times higher than that of the legitimate sources, even if routers cannot distinguish between legitimate and attack packets. The figure also shows that without packet prioritization, the admission ratio of legitimate packets decreases as the attack strength grows. This result shows the effectiveness of using static up-path and packet prioritization in STRIDE.

3.5.2 Flow Bandwidth Guarantees

STRIDE's bandwidth guarantees effectively isolate the bandwidth of attack traffic from that of legitimate traffic. As a consequence, in STRIDE, the effects of attacks are confined within the paths they follow regardless of whether attack sources flood a single path (or a link) or multiple paths simultaneously. We show this bandwidth isolation via large-scale simulations. For realistic simulations, we construct simulation topologies using a CAIDA SkitterMap [1], attach 10,000 legitimate sources to 200 ADs proportional to the AD size, and attach attack sources (hosts) to 100 ADs. Paths are probabilistically sampled from the SkitterMap to satisfy both the number of sources and the number of ADs. Legitimate sources control their packet sending rate based on the TCP congestion control mechanism, while attack sources send constant, high-rate traffic to flood a target link. We increase the attack size from 10K to 100K to compare STRIDE's bandwidth guarantees with those of a per-flow fair-sharing based mechanism. We consider a baseline case, labeled as "No Defense", where packets are randomly dropped during congestion.

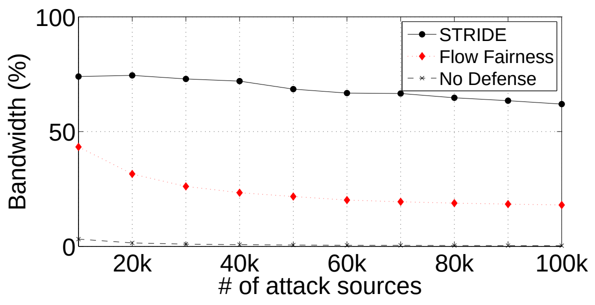


Figure 3.5: The impact of attack size.

Figure 3.5 shows the bandwidth used by the legitimate flows that originate from clean ADs. Under "No

Defense", the legitimate flows obtain almost no bandwidth. DDoS attacks. When per-flow fair-sharing bandwidth control is employed, attack flows cannot completely exhaust the target's link bandwidth, yet the attack effects grow linearly with the attack size.

STRIDE provides consistent bandwidth guarantees to legitimate traffic under different attack sizes, which proves the effectiveness of path bandwidth isolation. The bandwidth of legitimate flows decreases slightly as the attack size grows for two reasons. First, the extra bandwidth that is not fully used by some paths (due to the TCP congestion control) is shared by other flows. Second, as the number of contaminated ADs increases, the number of clean ADs decreases (as the total number of ADs is fixed).

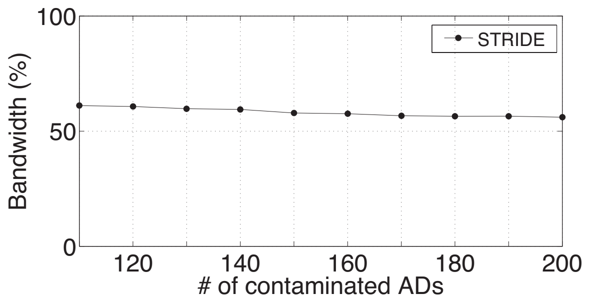


Figure 3.6: The impact of attack dispersion.

Next, we increase the number of contaminated ADs by 10 up to 200 ADs. As one can imagine, the bandwidth of legitimate flows decreases as Figure 3.6 shows. However, the effects of attack dispersion are marginal (i.e., proportional to the number of attack ADs) because the dynamic channel bandwidth is proportional to the static channel bandwidth and the static channel bandwidth that can be used by attack traffic is limited by the number of attack ADs in STRIDE.

3.5.3 Throughput

STRIDE introduces additional computational work for capability (or opaque field) verification. To gauge the computational overhead, we measure the throughput of a STRIDE router for various packet sizes and compare the result with that of the default IPv4 forwarding. We implement a STRIDE router as a user-space process using the Click Modular Router [89]. The capability generation and verification are implemented as CBC-MAC with AES-ni. We perform the measurement with a simple topology where a source and a destination are directly attached to a STRIDE router. Netperf[3] is used for throughput measurement.

[3]Netperf Benchmark. http://www.netperf.org/netperf/

As described earlier, STRIDE forwards packets based on the interface identifier in the packet header. Hence, unlike in today's routers, no overhead will be incurred for FIB (forwarding table) lookup. Meanwhile, the IPv4 forwarding in our experiments would produce the highest throughput that it can achieve since the FIB has only one entry in our network configuration.

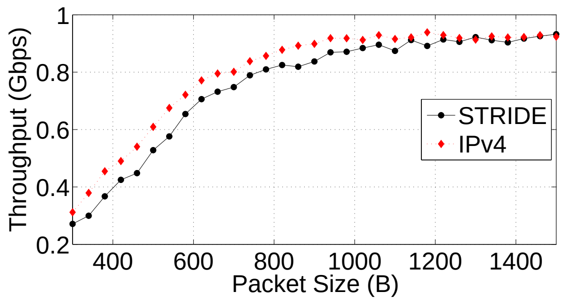


Figure 3.7: Throughput vs. packet size.

Figure 3.7 shows that for small packets, both IPv4 and STRIDE routers under-utilize the link bandwidth while the IPv4 packet forwarding outperforms that of STRIDE. For large packets, they both utilize more than 90% of the link bandwidth. In practice, the packet overhead becomes negligible because small packets account for less than 10% of bandwidth and most of remaining packets are full sized [1].

3.6 Extensions

This section presents two extensions to enhance STRIDE's waiting time guarantees. The first one extends the guarantees to inter-ISD communication. The second one strengthens the guarantees for requests using the BE channels on the down-paths.

3.6.1 Inter-ISD Guarantees

While STRIDE focuses on domain-based guarantees for communication within an ISD, many of the properties also translate for communication between ISDs. For example, static channels on the up-paths still guarantee low-capacity throughput. Only BE channels are getting lower guarantees, as there is no explicit indication that the receiver desires the communication. Thus, establishing a connection to a public service that is under attack will be challenging for an external host. However, as soon as the service receives one initial packet and desires to serve that client, it can set up a dynamic channel with the same protected bandwidth guarantees within each ISD assuming no congestion on high-capacity links between ISDs.

STRIDE gives precedence to internal static traffic over external static traffic because external static traffic may be labeled incorrectly per up-path. For example, a malicious AD or ISD can claim to have more capacity on a static path than the actual allocation, and other ISDs cannot verify the claim. Therefore, ISDs need to be able to monitor and limit external static traffic by itself (rather than trusting other ISDs) so as to provide connection setup guarantees to external hosts.

In this extension, ISDs isolates external traffic via explicit allocation. To achieve this goal, we consider three methods, each of which provides an ISD with different control granularity on external traffic.

1. *A common limit on all down-paths via Inter-ISD agreements.* Tow ISDs negotiate bandwidth limits in their contractual agreement. For example, ISD A can agree to allocate 1% bandwidth of each static path and 1% BE-class bandwidth on each link to ISD B's traffic that ISD B claims to be from static up-paths. ISD Cores and ADs at the ISD boundary are responsible for monitoring and regulation.

2. *Destination-controllable bandwidth limits via path publishing.* When publishing a path to the Path Server, the destination AD can specify how much bandwidth to allocate for external traffic on a per-path basis. The ISD Core encodes the destination-specified limit to the path, and performs monitoring and regulation accordingly.

3. *ISD-specific via path activation.* The destination AD constructs special BE or static down-paths dedicated to forward traffic originating from certain ISDs. Specifically, each AD on the path encodes the list of allowed ISDs into the opaque fields. Every on-path AD knows the policy and thus can perform regulation. Compared to the second method, this method is more flexible (i.e., this one does not depend on the ISD Core for regulation) yet requires more protocol rounds to establish such special paths.

3.6.2 Partial Dynamic Channels

A request using a static up-path and a BE down-path may be dropped by any congested ADs on the BE down-path. This leads to at least two disadvantages. First, the probability of reaching the destination decreases exponentially with the number of bottlenecks (i.e., congested ADs) on the BE down-path because these congested ADs drop packets randomly without any coordination. Second, it is a waste of work for the ADs on the previous hops, as they have to recycle allocated dynamic bandwidth due to the incomplete requests.

To overcome these two disadvantages and strengthen the guarantees when using BE down-paths, we propose to use *extensible dynamic channels* by extending the incomplete dynamic channel established between the sender and the AD that drops the request.

The sender resends requests on an incomplete dynamic channel to extend the incomplete dynamic channel until either it reaches the destination or the channel expires. In this way, the expected waiting time is reduced from multiplicative to additive with respect to the number of congested ADs. Moreover, since the incomplete dynamic channel is used only for sending requests, ADs on the path can reduce the allocated bandwidth for such incomplete channels for improved bandwidth utilization.

By contrast, prior work providing per-flow or per-source fairness at each bottleneck independently may suffer from low link utilization in the case of multiple bottlenecks. For instance, consider a topology with links A-B (2 units of bandwidth), B-C (1 unit) and B-D (1 unit). There are 99 one-unit flows from A to C and a one-unit flow from A to D. While the topology can deliver 2 units of traffic end-to-end, a per-flow fair share policy would degrade the utility from 2 to 1.02.

3.7 Discussion

3.7.1 Malicious ADs Inside an ISD

ADs within an ISD may get compromised, therefore failing to regulate traffic. Although this is unlikely in well-administered ADs, attackers can nevertheless exploit software vulnerabilities in routers or administrative workstations. STRIDE can identify malicious ADs using neighborhood monitoring and existing fault detection protocols [180]. For example, if any AD sends more traffic than their allocated share, the AD must be malicious or misconfigured, and the neighboring AD can block the offending traffic. As a technical defense, once the malicious AD is detected, hosts can avoid the malicious AD by selecting paths that avoid traversing that AD. Most importantly, since all ADs of an ISD are within a uniform legal environment, the ISD Core can revoke the membership of misbehaving ADs.

3.7.2 A Simple Dynamic Allocation Policy

As an example, we describe a simple dynamic allocation policy that satisfies two requirements: (1) every request traversed a static path can obtain at least a certain amount of allocation that is known a priori, and (2) the sender can always renew an established dynamic channel (i.e., extending the expiration time of the channel) by sending a renewal request using the current dynamic channel. The capacity of the renewed channel may be less than the old one but is still lower-bounded by a certain value that is known by the sender prior to sending the request.

We consider a simple scenario where all dynamic requests are homogeneous, i.e., with the same length t and the same bandwidth amount b. The static bandwidth of a link is one unit with a request rate limited to r, and the dynamic bandwidth is D units. To ensure that every request on the static path can successfully make a reservation, it is required that $D \geq r \cdot b \cdot t$.

To support fast renewal using the dynamic channel (where the sender sends renewal requests using the established dynamic channel), the bandwidth estimation for D has to be revised based on the renewal policy. Here we consider a simple renewal policy: let x be the amount of reserved bandwidth that will be freed (and may be renewed) during the current time interval. If $x \leq \frac{D}{2t}$, then the renewable bandwidth of a flow is the same as the previous amount. Otherwise, the renewable bandwidth of a flow is a half of the previous amount. In this case, by setting $D \geq 2r \cdot b \cdot t$, we ensure that every request on the static path can reserve at least b bandwidth for t time, and every renewal request can get at least a half of the previous amount.

This policy ensures that the total allocation never exceeds D at any point of time because the newly reserved bandwidth never exceeds $\frac{D}{2t}$ for any time interval and so does the bandwidth renewed during this interval.

A similar argument can be applied to configure the BE bandwidth. If the BE bandwidth of a link can forward requests at rate up to r_{be}, then it is required that $D \geq 2(r + r_{be}) \cdot b \cdot t$.

Note that although the sender could renew more than once and get multiple capabilities at hand, all the updated capabilities would be associated with the same flow ID. Hence, they can be correctly monitored and regulated.

3.7.3 Comparison with Other Bandwidth Reservation Protocols

A major difference between STRIDE and other bandwidth reservation protocols (e.g., RSVP [177]) is that other approaches are not designed with security in mind and thus fail to consider two important guarantees in the presence of DDoS adversaries:

1. Waiting time guarantees: When will the reservation request be granted? During DDoS attacks targeting the requests (e.g., Denial of Capability attacks), the sender may never be able to deliver the request to the receiver, thereby rendering infinite waiting time.

2. Bandwidth guarantees: How much can the sender reserve?

On the other hand, STRIDE aims to secure the reservation requests from DDoS and also provides lower bounds on reservable bandwidth.

Besides providing diverse guarantees that prior work does not achieve, STRIDE is more scalable compared to prior work. For instance, the bandwidth allocation mechanism in STRIDE is more scalable than the RSVP protocol used by the QoS Integrated Services (IntServ), as RSVP requires the sender (which can be a host or an AD, when RSVP aggregation is used as RFC 3175 specifies) to make an end-to-end reservation to the receiver(s) causing a quadratic number of control messages (in the number of entities) in the network and quadratic state on the intermediate routers. In STRIDE, since control messages are sent along a tree-like topology, the control messages and state are linear to the number of ADs.

Another advantage of STRIDE is flexibility. End-to-end reservation (e.g., circuit-switch networks) is typically less flexible than statistical multiplexing (e.g., packet-switch networks). Interestingly, by supporting diverse bandwidth classes and path splicing, STRIDE can be viewed as a combination of packet-switch and circuit-switch and therefore obtain the best of both worlds.

3.7.4 Directional Paths and Asymmetric Bandwidth Requirements

In practice, network links may be directional or asymmetric with different bandwidths in the two directions. STRIDE can flexibly accommodate asymmetric paths with minimal modifications as follows. To request a packet on a directional path, the source puts in the header both the forward and backward paths. For example, dynamic channels can be requested on both the forward and return paths, and sent back to the other party through sufficient space allocated in the packet header. Bandwidth requests may be asymmetric, as in downloads the client-to-server bandwidth is one to two orders of magnitude smaller (acknowledgment packets are smaller than data packets). In this case, STRIDE supports asymmetric bandwidth allocations.

3.7.5 Link Capacity Division

Using estimations, we provide guidelines that an AD can follow to divide its total link capacity to three traffic classes—static, dynamic, and BE. First, given that the current real-world link utilization is mostly below 30% based on the CAIDA dataset [1], allocating 30% of the link capacity to the BE class would satisfy legacy Internet traffic in most cases. Subsequently, assuming the static and dynamic classes are allocated s and d fractions of the link capacity, respectively, the following conditions should hold:

$$s + d = 1 - 30\% \tag{3.1}$$

$$40\text{Gbps} \times s > 500\text{Kbps} \times 10000 \tag{3.2}$$

The first condition ensures a link will not be overloaded when each bandwidth class is being fully utilized. The second condition assumes an OC-768 link capacity (40 Gbps) to be divided among around 10000 paths, such that each endpoint in a medium-size ISD (e.g., a US ISD with around 2200 ADs, according to the CAIDA dataset) can choose up to 10 paths in our experiment. The second condition requires the static bandwidth allocated to each path be no less than 500 Kbps.

Based on these guidelines, a reasonable example allocation is to divide 5 – 15%, 60 – 65%, and 30% link capacity to the static, dynamic, and BE traffic classes, respectively. In practice, an AD can adjust the numbers in the conditions based on its own link capacity, number of current paths that the AD supports, etc. Furthermore, when any bandwidth class is not fully utilized, other congested traffic class can take up all the bandwidth that is currently available.

3.7.6 Bandwidth Overbooking

Section 3.3.1 introduces bandwidth overbooking for simultaneous enhancement of path quality and diversity but with possible denial of path activation. To mitigate this issue, we suggest an appropriate overbooking ratio by analyzing the relationship between an overbooking ratio and the corresponding probability of path activation denial as follows.

We consider an intermediate AD AD_p wanting to determine its overbooking ratio. Let I_i and E_j represent the i^{th} ingress interface from the providers ($0 \leq i \leq l$) and the j^{th} egress interface to the customers ($0 \leq j \leq m$), respectively. Let each ingress interface connect to all m egress interfaces. We assume that m interfaces connect to n customer ADs (i.e., each customer AD has $\frac{m}{n}$ links to AD_p). Then, each customer AD has at least $\frac{l \cdot m}{n}$ distinct paths to the ISD Core through AD_p. In this setting, suppose each customer AD selects uniformly at random k out of the $\frac{l \cdot m}{n}$ paths to the ISD Core, then the probability that the customer ADs select I_i more than t times in total would be: $P_{I_i}(t) \approx 1 - \sum_{i=0}^{t} e^{-\lambda} \cdot \frac{\lambda^i}{i!}$, where $\lambda = \frac{n \cdot k}{m}$. This implies that I_i's bandwidth needs to be allocated to t egress interfaces (out of m interfaces), which would increase per-path bandwidth allocation by $\frac{t-\beta}{\beta}$, where $\beta = \frac{n \cdot k}{l}$ is the average number of activated paths through I_i. If $t \gg \frac{n \cdot k}{l}$, sufficient path diversity (as much as $\frac{t \cdot l}{n \cdot k}$) is provided to customer ADs.

As a result, AD_p may determine t such that the probability of the denial of path activation does not exceed some threshold P_{th} (i.e., $P_{I_i}(t) \leq P_{th}$). For example, P_{th} = 0.2 means that 80% of path activation requests would be accepted on average. Hence, the expected number of trials for successful path activation becomes 1.25. That is, P_{th} determines the number of requests that should be made by an endpoint AD until successful path activation.

3.8 Summary

A core goal of the STRIDE architecture is to achieve intrinsic DDoS defense with relatively simple routers. In particular, we avoid per-flow state in the fastpath, asymmetric cryptographic operations, reliance on untrustworthy domains, and key establishment across ADs. Even with our relatively simple operations, we can achieve protection against DDoS from large botnets. Reflecting on the STRIDE architecture, we observe that measured trust in ADs located within the same legal environment providing viable prosecution helps to simplify the architecture and results in higher efficiency, meanwhile the untrustworthy ADs outside the trust domain cannot inflict damage against local within-trust-domain communication. We anticipate that STRIDE provides a useful point in the design space to study holistic network architectures with strong DDoS defense properties.

Chapter 4

RainCheck Filter

Chapter 3 presents a DDoS-limiting architecture that provides (1) probabilistic waiting time guarantees for accessing public services and (2) bandwidth guarantees to established flows. This chapter explores how to achieve stronger deterministic waiting time guarantees—Maximum Waiting Time guarantees—to overcome a bottleneck on the communication path. Combined with STRIDE, such a primitive can be used to provide maximum waiting time guarantees to access public services. We discuss the integration in detail in Chapter 7.

For ease of presentation, we use server flooding (i.e., the bottleneck is at the server) as a motivating example throughout this chapter.

Motivation: Challenges of providing waiting time bounds in the face of DDoS attacks. Internet users are impatient. A recent study found that more than half of online shoppers abandon websites that fail to load in three seconds [3]. When a wait is unavoidable, users perceive known, finite waits to be shorter than uncertain waits [113], and are willing to wait much longer periods if given visual feedback, such as a progress bar [124, 24].

In the presence of Distributed Denial of Service (DDoS) attacks, users would suffer from an uncertain or even infinite waiting time for accessing an online service, as neither the server nor the user knows when the attack will cease. Unfortunately, DDoS attacks are easy to launch and can cause severe damage. For example, buying access to 10,000 compromised hosts costs only $200 [42], whereas one-minute unavailability costs an average of $22,000 [132]. Even worse, DDoS attacks are hard to defend against. Enterprise solutions for server DDoS mitigation (e.g., adding more servers or using content delivery networks) may be too costly for small- and medium-sized companies to afford. Also, sophisticated DDoS attacks successfully emulate flash crowds; it is therefore difficult, if not impossible, to differentiate attack traffic from legitimate traffic for DDoS defense. Resource-based proof-of-work proposals can provide each client with a fair share of resources, but are impractical since they consume critical computational or bandwidth resources of clients, while giving an unfair advantage to resourceful hosts [163, 129].

Furthermore, none of the aforementioned defense mechanisms can provide a Maximum Waiting Time (MWT) guarantee such that a client's request is accepted for service within a finite time T. Gligor [67] shows how to achieve MWT using a rate-control service that performs precise request scheduling at line rate. However, such precise scheduling may be too complex to implement or may result in low server utilization if the scheduled requests do not appear.

Our solution: Bounded waiting time to get through a bottleneck. In this chapter, we present RainCheck Filter, a DDoS mitigation primitive that guarantees MWT when a critical resource, such as the computational power or network bandwidth of a server, is exhausted. The core idea behind RainCheck Filter is simple yet effective: RainCheck Filter prioritizes clients' service requests based on how long each one waited, as if the service owning the critical resource maintains an infinite queue in which the clients' requests are lined up waiting for the service. To simulate the infinite queue with a small physical buffer, the service sends the client a *raincheck*,[1] a timestamped cryptographic token which not only tells the client when to retry but also serves as a proof of the client's priority level. Namely, we approximate an infinite queue at the service using rainchecks, network propagation delay, and clients' buffer—which collectively constitutes a *virtual queue*. Rainchecks are only valid for a limited time duration so that the resource can efficiently rate limit each client and prevent raincheck reuse without keeping per-client state. RainCheck Filter can be applied in front of a critical resource, for example, as a middlebox for a flooded link or a server. For the rest of this chapter, we assume that RainCheck Filter is applied to protect the critical resource at the server.

We prove that in RainCheck Filter a legitimate client can access the server within a finite time T which is linear with the number of clients. We demonstrate how to strengthen the waiting time guarantees by exploiting auxiliary information such as distinguishability between bots and legitimate clients and the non-uniform distribution of bots. To provide users with information on their actual waiting time, we devise a server-side algorithm that accurately estimates each client's waiting time. The server refines the estimates over time by incorporating the current client status.

Besides achieving strong guarantees, RainCheck Filter is also flexible, lightweight, and easy to deploy. RainCheck Filter can be used at different network protocol layers, at different networking elements, and in conjunction with other DDoS countermeasures. It neither exhausts scarce bandwidth or CPU resources,

[1]A raincheck is an idiom that represents a promise that an unaccepted admission will be renewed in the future.

nor does it require any special networking infrastructure. Rainchecks can be seen as a type of a *cookie* (e.g., TCP SYN cookies [104]) since they free the server from keeping the client information, thereby avoiding memory-based resource exhaustion. On the other hand, a raincheck is more than a cookie, as the encoded timestamp allows the server to bound the client waiting time in the face of any form of server flooding.

Our main contributions are as follows:

- We present RainCheck Filter, a lightweight DDoS mitigation primitive that helps legitimate clients obtain their fair share of the server's processing power by utilizing the network as an infinite virtual queue.
- We prove that RainCheck Filter achieves MWT guarantees without per-client state on the server or precise request scheduling, which none of the prior work can achieve.
- We propose a rank estimation algorithm that accurately estimates each client's actual waiting time based on a probabilistic counting technique.
- We evaluate RainCheck Filter using theoretical analysis, simulations, and an implementation. Our simulation indicates that RainCheck Filter can also reduce variance in the waiting time. Such a characteristic empowers servers to provide clients with reliable feedback.

4.1 Problem Definition

Our goal is to provide Maximum Waiting Time (MWT) guarantees for clients to get through a bottleneck on the communication path. For ease of presentation, we focus on the server flooding scenario, where the attacker depletes the server's scarce resource (e.g., processor, disk I/O, or internal bandwidth). Our solution can also be applied to address link flooding, where the attacker depletes the bandwidth of network links, such as access links to servers.

4.1.1 Waiting Time Model

Among various waiting time guarantees, we consider a strong notion called *maximum waiting time (MWT) guarantees*—a finite time T within which a client's request is accepted for service [67].

From the server's perspective, the waiting time of a client request c that is accepted after r_c retries is $T(c) = T(c, r_c) - T(c, 0)$, where $T(c, i)$ is the time at which the server sees the ith retry by client c, and the 0th try represents the original request. Similarly, the waiting time observed by a client, $T'(c)$, is the time elapsed between when the client first sends the response and when the client receives the response, and $T'(c) = T(c) + RTT(c) + process(c)$, where *RTT(·)* and *process(·)* indicate the round trip time and the server's request processing time, respectively. Assuming *RTT(·)* and *process(·)* are bounded, a bounded $T(c)$ implies a bounded $T'(c)$. Hence, without loss of generality, we consider only the waiting time observed by the server.

To support this model, we assume that clients and servers are loosely time synchronized (within a few milliseconds) using standard protocols such as NTP.

4.1.2 DDoS Attacks and Flash Crowds

In server flooding, the server's scarce resource is depleted by requests[2], thereby increasing the waiting time for legitimate clients.

In a flash crowd, the server is swamped with requests from legitimate clients alone. In a bot-driven DDoS attack, the adversary directs compromised endhosts, or bots, to overload the server with an overwhelming number of requests. Bots can forge their IP addresses to evade IP-based detection. We make no assumption on the adversary's power and strategy. For example, a powerful and smart adversary can compromise the majority of endhosts and target one client to maximize the client's waiting time.

Since this chapter focuses on guaranteeing MWT in the face of server flooding (e.g., consuming the server's computational resources), we assume a congestion-free network with benign routers. RainCheck Filter can also handle link flooding with a single congested link (e.g., flooding an access link in front of the server), where the ISPs or routers in front of the congested link issue rainchecks. In the rare situation in which multiple components in the network are flooded (e.g., multiple links, or a link and a server), then existing mitigation approaches for link flooding can be applied [111, 102]. For detecting malicious routers that manipulate traffic (e.g., dropping or delaying packets), existing fault localization techniques can be applied [16, 69, 49, 179].

4.1.3 Server and Client Models

Server model. When a request arrives, the server first performs some operations at line rate, such as replay detection. Since the incoming request rate, R_{in}, is bounded by the network line rate, the server can process every request before adding it to a queue. The queue is kept in fast memory such that the enqueue and dequeue operations can be done at line rate as well. The server's processing rate, R_s, is limited by bottleneck operations (e.g., slow password derivation or database query). When $R_{in} > R_s$, the queue overflows and drops requests based on a certain policy such as tail drop.

[2]For example, many servers adopt iterated password hashing (e.g., the PBKDF2 algorithm with 1000 iterations) to slow down password cracking to the order of milliseconds per password [4]. Due to such slow password derivation, the server can be flooded by simply hundreds of login requests per second.

Client model. We consider a network of N clients, consisting of Z *compromised* and $N - Z$ *legitimate* clients, where $Z \leq N$ and N are bounded but may be unknown to the server. Typically, compromised clients are controlled by the adversary via malware, while legitimate clients are controlled by their human users. Each client is attached to an endhost machine and has a unique identifier. We begin with a simplified case where each client has a unique IP address and discuss in Section 4.2.5 a generalized case where clients may share one IP address because they are machines behind a Network Address Translator (NAT) or are attached to the same machine.

Legitimate clients can be either greedy or non-greedy. A greedy client seeks to minimize its waiting time without raising any suspicion by exploiting some loophole in the protocol specification. A non-greedy client strictly complies with the communication protocols.

We consider flooding by initial requests but not handshake messages or data following the requests, as the server can rate limit the requests and never accept requests at a rate higher than what it can support.

4.1.4 Desired Properties

MWT guarantees. The DDoS-limiting primitive should bound the waiting time of a legitimate client, and the bound should be independent of other clients' strategies.

Accurate feedback. The server's estimate of a client's waiting time should be within a reasonable error margin of the actual waiting time in order to increase users' willingness to wait.

Minimal overhead for both clients and servers. The DDoS-limiting primitive should incur minimal overhead for both servers and clients, thereby avoiding the increase of the attack surface. In particular, the primitive should avoid per-request or per-client state on a server.

4.2 Raincheck Filter

Our core observation about RainCheck Filter is that Maximum Waiting Time (MWT) guarantees can be achieved if the server keeps a large queue of size N, where N is the number of clients. We call this an ideal buffer because in reality we would like to avoid keeping per-client state. RainCheck Filter simulates the ideal buffer using a *realistic buffer* whose size is much smaller than N by *leveraging the network as an infinite virtual queue*. This is achieved through the exchange of a special type of message called *raincheck* between the client and the server.

We first present a simple approach using an ideal buffer and discuss its fundamental properties that help achieve MWT guarantees. We then present RainCheck Filter, which satisfies these fundamental properties, but with a realistic buffer. For the sake of simplicity, we assume that each client has a unique IP address in Section 4.2.1– 4.2.4 and show how RainCheck Filter works when multiple clients share one IP address in Section 4.2.5. The notation is summarized below.

c	Client ID (e.g., IP address)
N	Total number of clients
Z	Number of compromised clients
L	Server queue length ($L \ll N$)
ρ_c	Client c's raincheck
R_s	Server's request processing rate
Δ	Raincheck expiration period
$TS(\rho_c)$	Timestamp encoded in ρ_c

4.2.1 MWT Guarantees Using an Ideal Buffer

Without any DDoS countermeasure, an attacker who sends a large number of requests has an advantage and a legitimate client has a disadvantage. While rate-limiting schemes can ensure fair resource allocation among clients, rate limiting itself is insufficient to achieve strong waiting time guarantees.

Using a buffer of size N, we can achieve MWT guarantees as follows: the buffer is modeled as a FIFO queue that euqueues incoming requests. By limiting each client to have no more than one request in the queue, we ensure that the buffer never overflows. Since the server can process R_s requests per time unit, the waiting time is bounded by $\frac{N}{R_s}$. While such an ideal case requires no IP spoofing as an assumption, RainCheck Filter can prevent IP spoofing by interacting with clients, as malicious clients sending DDoS packets with spoofed addresses will not receive the returned rainchecks, as we describe later.

This approach adopts a simple rate-limiting policy (i.e., one request per client in the queue) as well as a request-ranking policy that orders requests by their age, or the time during which a request has stayed in the buffer. The server processes the request with the lowest rank (i.e., the oldest request) first.

We observe that the request-ranking policy presents two properties that lead to MWT guarantees in this ideal case: (1) the initial rank (in the virtual queue) of each request is bounded, and (2) the rank decreases over time. In Section 4.3.1, we generalize this observation and present a theorem that we use as a guideline to design RainCheck Filter and prove its MWT guarantee.

4.2.2 RainCheck Filter Design

With a buffer of size $L \ll N$, a flooded server has to discard most of the requests, which makes it difficult to treat each client fairly and bound the waiting time.

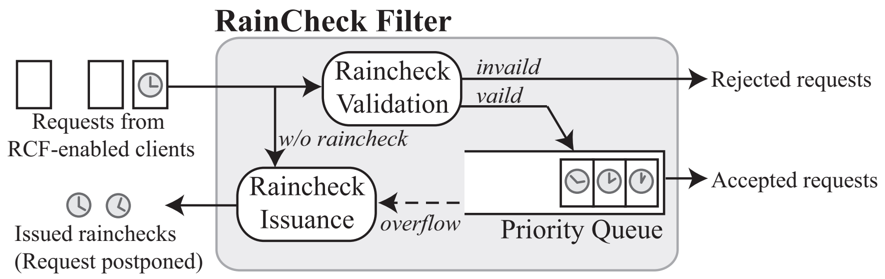


Figure 4.1: RainCheck Filter overview.

To address this challenge, RainCheck Filter leverages the network as an infinite virtual queue from which the server can retrieve the knowledge of previously dropped requests. In particular, rather than silently dropping a request, the flooded server sends to the client a cryptographic token called *raincheck*, with which the client can prove how long it has waited to be served. RainCheck Filter effectively simulates the ideal buffer, thereby resulting in a bounded waiting time without keeping per-client state.

Overview. Figure 4.1 illustrates how RainCheck Filter works on the overloaded server. The server can (1) accept, (2) reject, or (3) postpone an incoming request, which could be either raincheck-carrying or raincheck-absent. When *postponing* a request, the server asks the client to revisit at a later time by issuing a new raincheck to the client. Note that a raincheck-carrying request can be postponed with a new raincheck. To manage rainchecks, RainCheck Filter implements two core components on the server side—*raincheck issuance* and *raincheck validation*—both of which operate at line rate.

The raincheck validation component checks the validity of a raincheck using the server's secret key. Requests with an invalid (e.g., expired) raincheck are rejected, while requests with a valid raincheck are added to a priority queue of length L, where a request that waits longer gets higher priority.

For each valid yet dropped request, the raincheck issuing component constructs a raincheck using a server's secret key and returns the raincheck to the client. Rainchecks are protected using Message Authentication Codes (MACs) to prevent forgery, tampering, or raincheck sharing. The client can resend its request with the associated raincheck as a proof of the waiting time.

Raincheck-absent requests are forwarded to the raincheck issuance component directly (rather than being assigned the lowest priority) for two reasons: (1) to prevent the server from queuing requests with spoofed IP addresses, since the client must return with the raincheck in order to use the server's service, and (2) to ensure bounded waiting time; otherwise, a raincheck-absent request can be stuck in the queue forever when raincheck-carrying requests arrive at the same speed as the server's processing rate.

One major challenge is how to ensure fair use of rainchecks in an efficient manner. Particularly, the server should prevent double-spending of rainchecks and rate limit the number of rainchecks consumed by each client without keeping per-client state. To address this challenge, we impose on every raincheck an expiration period Δ, which is shorter than the time it takes to process all client requests. Consequently, the server only needs to keep requests accepted in $[t - \Delta, t)$, where t is the current time. Clients wishing to remain in the virtual queue have to periodically renew their rainchecks. The server can adjust Δ to strike a balance between communication and storage overhead, as discussed in Section 4.3.2.

Raincheck message format. A raincheck contains a Message Authentication Code (MAC) protecting the client ID c, timestamp ts, and the lifetime $[t_{start}, t_{end})$, computed with the server's secret key k, such that the adversary cannot tamper with or forge a raincheck:

$$\rho_c = m \| MAC_k(m), \text{ where } m = c \| ts \| t_{start} \| t_{end}. \tag{4.1}$$

A unique client ID c is included to enable rate limiting based on source identities and also to prevent two clients from sharing their rainchecks. Since each MAC is keyed using the server's secret key, only the server can correctly create and validate the raincheck.

4.2.3 Server Description

Raincheck issuance. When a queue overflows, the raincheck-carrying request is dropped and directed to the raincheck issuance component for renewal. The server renews the raincheck by updating its lifetime: the timestamp stays the same as the one in the old raincheck ($ts = t_{old}$) but the raincheck is valid from $t_{start} = cur\_time + t_{pause}$ to $t_{end} = cur\_time + \Delta$, where cur_time is the current time and t_{pause} is a small amount of time indicating how long the client has to wait at least before resending. When a raincheck-absent request arrives, the server drops the request directly and returns to the client c a raincheck in which the timestamp is the current time ($ts = cur\_time$) and the lifetime is $t_{start} = cur\_time + t_{pause}$ and $t_{end} = cur\_time + \Delta$.

Note that t_{pause} does not affect our MWT guarantees; the guarantees are independent of clients' resend strategy, as proved in Section 4.3. Section 4.4.2 explores a RainCheck Filter variant in which rainchecks can have different lifetimes for improved scalability.

Since the server issues a raincheck to every dropped request, a client can have multiple valid rainchecks concurrently. However, having multiple valid rainchecks provides no additional benefits to the client, because the server (or its raincheck validation component) is designed such that the acceptance of a client's request invalidates all but a limited number of rainchecks that the client has.

Raincheck validation. For efficient double-spending prevention and rate limiting, the server keeps a set *Accepted* that contains requests that were accepted during time $[t - \Delta, t)$. We denote by $Accepted(c)$ whether a client c's request is in the set. Similarly, we denote by $Buffered(c)$ whether c's request is currently buffered in the queue. The server can implement the checks $Accepted(c)$ and $Buffered(c)$ efficiently using a Spectral Bloom Filter [35] or a Sliding Bloom Filter [125].

A raincheck is valid if all of the following conditions hold:

1. **Lifetime.** $t_{start} \leq cur\_time < t_{end}$.

2. **No duplicate.** The same raincheck cannot be reused more than once.

3. **Limited client request rate.** Given the limit n determined by the raincheck validation component, when a client's raincheck is accepted, at most $n-1$ additional unique rainchecks can be renewed or accepted for the same client during Δ. In the case when $n = 1$, the server should not have recently accepted or queued the client's request within Δ (i.e., $Accepted(c) = False$ and $Buffered(c) = False$). In addition to rate limiting via per-source fairness, RainCheck Filter can also work with other fairness models to accommodate clients sharing the same IP address (e.g., clients behind a NAT). We address this issue in Section 4.2.5.

4. **Integrity.** MAC verifies correctly.

The first three conditions ensure that once a request is accepted, all but $n-1$ rainchecks that the client has at that point become invalid.

Valid requests are added to the priority queue. If there are already L requests in the queue, the lowest priority request will be dropped. The server ranks requests unambiguously based on their timestamps.[3] For the rest of the chapter, we assume that the raincheck validation component allows one raincheck per client ($n = 1$).

In practice, requests or rainchecks may be lost without any malicious actions. To limit the impact of packet loss, the server informs a client of the acceptance of the client's raincheck before processing the request, such that the client can retry if it does not hear back from the server after certain time. Since the acceptance notification may be lost too, the server will notify the client of the acceptance again if the client has recently been served (i.e., $Accepted(c) = True$).

4.2.4 Client Description

Figure 4.2 demonstrates the client-server interaction in the RainCheck Filter protocol. The client initiates the RainCheck Filter protocol by sending a raincheck-absent request. The server returns to the client a raincheck that expires after Δ in the future.

[3]A trade-off exists between the timestamp granularity and the waiting time bound. When at most v requests are allowed with the same timestamp value, the waiting time bound is increased by $\frac{v}{R_s}$.

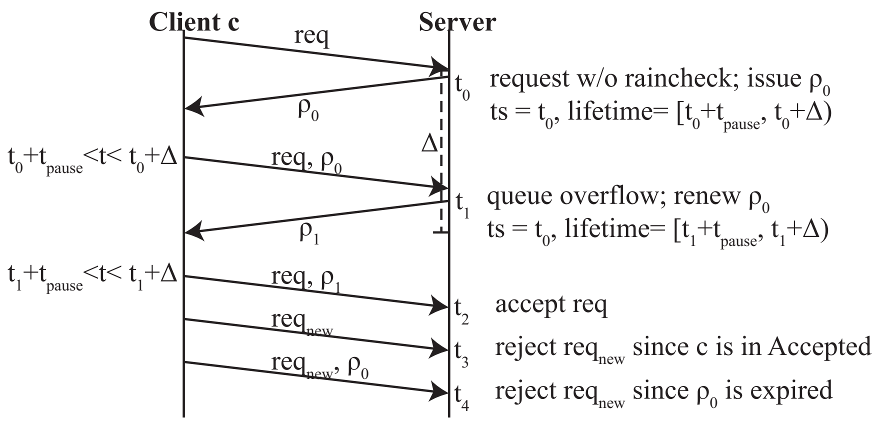

Client c
Server
req
t_0
request w/o raincheck; issue ρ_0
ρ_0
ts = t_0, lifetime= $[t_0+t_{pause}, t_0+\Delta)$
Δ
$t_0+t_{pause}<t<t_0+\Delta$
req, ρ_0
t_1
queue overflow; renew ρ_0
ρ_1
ts = t_0, lifetime= $[t_1+t_{pause}, t_1+\Delta)$
$t_1+t_{pause}<t<t_1+\Delta$
req, ρ_1
t_2
accept req
req_{new}
t_3
reject req_{new} since c is in Accepted
req_{new}, ρ_0
t_4
reject req_{new} since ρ_0 is expired

Figure 4.2: Client-server interaction example.

The client resends the raincheck-carrying request before the raincheck expires. If the retry fails, the client obtains a renewed raincheck with an extended lifetime. The client keeps resending until the request is accepted. We prove in Section 4.3 that a client following this resend strategy will be able to access the server after a bounded time.

While a greedy client may attempt to reduce the waiting time by resending the request as quickly as possible, it does not help to send at a rate faster than $\frac{R_s}{L}$ because the buffer keeps only one copy of the request for each client. Moreover, the server explicitly specifies in the raincheck how long the client has to pause before retrying (i.e., setting t_{start} in the raincheck's lifetime) to further restrain greedy clients and to minimize communication overhead.

4.2.5 Handling Clients Sharing IP Addresses

So far RainCheck Filter assumes that each client has a unique IP address and provides per-source fairness, which effectively limits the sending rate of requests originating from the same source IP address. In practice, per-source fairness may be unfair for clients sharing the same IP address, such as clients behind a Network Address Translator (NAT). We now relax the assumption by showing how RainCheck Filter can work with other fairness models that replace the rate-limiting check in the raincheck validation component.

RainCheck Filter is a general framework, and depending on the service type, the server can make differential allocations based on history, IP-prefix, IP, domains, etc. In this chapter, we use an IP-based policy model for illustration. The server can consider a fairness model where the allocation of server resources is proportional to the number of requests during peacetime (i.e., when the server is not flooded). Specifically, the server splits IP addresses into blocks and measures the number of requests served per address block during peacetime. During an attack, such information is used for a fair allocation of server resources based on the IP address space. This requires keeping state for each address block.

4.3 Analysis

We analyze RainCheck Filter's waiting time guarantees, the computational, communication, and storage overhead, as well as security benefits.

4.3.1 Waiting Time Guarantees

Properties of a rank function ensuring MWT. Let $t_{ini,c}$ be the time at which the server sees client c's first trial and $t_{acc,c}$ be the time at which the server accepts c's request/retry. We denote by $rank(c,t)$ client c's priority of service (e.g., the position in the ideal queue) at time t, and $rank(c,t)$ is defined only for $t_{ini,c} \leq t < t_{acc,c}$. Client c is served immediately at time t when $rank(c,t) = 0$.

Theorem 4. *There is a bound T such that for all c, $t_{acc,c} - t_{ini,c} \leq T$, and T is independent of the adversary's power or strategy if the rank function satisfies the following two conditions:*

1 *The initial rank of each client is bounded: $rank(c, t_{ini,c}) \leq B$ for all c, and B is adversary-independent.*

2 *The rank of each client decreases over time: $\exists\, \delta > 0$ and $\gamma > 0$ such that $rank(c, t-\delta) - rank(c,t) \geq \gamma > 0$ for all c and t, and δ and γ are adversary-independent.*

Proof sketch: Since it takes at most $\frac{\delta B}{\gamma}$ time to reduce a client's rank to zero, the waiting time is bounded: $T \leq \frac{\delta B}{\gamma}$. That is, the server guarantees MWT for any ranking function that satisfies the above two conditions. □

The rank function. The rank function in RainCheck Filter can be formulated as follows. We say c_1 has a lower rank than c_2 at time t, or $c_1 <_t c_2$, if at time t client c_1 has a valid raincheck whose timestamp is smaller than any of client c_2's valid raincheck. Hence, $rank(c,t) \triangleq |\{c' | c' <_t c\}|$. When $rank(c,t) < L$ (i.e., server queue length), c's request will always be accepted.

Theorem 5. *RainCheck Filter guarantees that a legitimate client will be served in a finite time T, regardless of how other (both legitimate or compromised) clients behave, and T is linear in the total number of clients.*

Proof sketch: In RainCheck Filter, the initial rank is bounded by N, the number of clients in the network. Second, between c's i-th and $i+1$-th retries that are at least $\frac{L}{R_s}$ time apart, the server either accepts c's request or accepts L requests from the more privileged clients. Also, RainCheck Filter ensures that once a client's request is accepted, all its rainchecks become invalid. Therefore, $rank(c,t)$ decreases by L for every retry, which means after at most $\lceil \frac{N}{L} \rceil$ attempts the server will accept the request. Also, as specified in Section 4.2, a legitimate client resends its request at a frequency f such that $\frac{1}{\Delta} \leq f \leq \min\{\frac{R_s}{L}, \frac{1}{t_{pause}}\}$. Hence, based on Theorem 4, the waiting time is bounded

$$T(C) \leq \lceil \frac{rank(C, t_{ini,c})}{L} \rceil / f \leq \lceil \frac{N}{L} \rceil \Delta. \tag{4.2}$$

□ When a client re-sends at a frequency $\frac{R_s}{L}$, it is guaranteed that its request will be accepted by the server in $\frac{N+L}{R_s}$ time. The waiting time $\frac{N+L}{R_s}$ (and the number of trials $\lceil \frac{N}{L} \rceil$) is also a lower bound in the presence of a strong attacker who knows the client's request sending schedule and controls every host except the victim client and server.

If the round trip time is not negligible compared to Δ, the bound should be revised to $\lceil \frac{N}{L} \rceil (\Delta + RTT)$ to compensate the delay.

If the client finds that the DDoS attack is not targeting the client, he can wait longer (e.g., Δ) before retry, as the bots will keep consuming their high-ranked rainchecks while the legitimate client waits silently. This demonstrates a strength of RainCheck Filter: the client can increase its priority by simply waiting, in contrast to prior work where the client has to "work", such as solving computational puzzles.

4.3.2 Overhead Analysis and Configurations

RainCheck Filter overhead. RainCheck Filter incurs low computational, communication, and storage overhead for both servers and clients. RainCheck Filter avoids keeping per-client state at the cost of sending rainchecks; a server can adjust the raincheck expiration period to strike a balance between communication and storage overhead.

Clients are not required to perform any additional computation. A client keeps the most recent raincheck for each server, and renews the raincheck roughly every Δ time period. A server has to perform one MAC generation for each issued raincheck and one MAC verification for each received raincheck. With an efficient MAC function, rainchecks can be generated and verified at line rate. The server stores requests that were accepted within Δ, which takes $O(\Delta R_s)$ space. Since each client renews its raincheck roughly every Δ time, the server sends and receives about $\frac{N}{\Delta}$ rainchecks per time unit.

Each raincheck is 32 bytes: 32 bits for the client IPv4 address, 64 bits for the timestamp up to microsecond granularity, and another 32 bits for the lifetime represented by the offsets using the timestamp as the reference time. The MAC is 128-bit CBC-MAC using the AES block cipher. To leverage hardware accelerated cryptographic feature, our prototype implements AES using AES-NI instruction set[4] supported by Intel processors and computing a MAC on a commodity server equipped with Intel i5-4430S takes only 61 cycles (equals to 22 ns). The server changes its secret key periodically, such that the attacker cannot recover the key in time. The raincheck is stored in the packet header, such as a custom HTTP header or a TCP option, as discussed in Section 4.6.

Parameter configurations. Let M be the size of the memory (in terms of the number of requests) available for RainCheck Filter at the server. Recall that R_{in} is the incoming request rate, R_s is the server's request processing rate, Δ the expiration period, and N is the number of clients.

According to Section 4.2, clients wishing to stay in the virtual queue must renew their rainchecks before they expire. Hence, the expiration period should be long enough to accommodate every client in the worst-case scenario, which leads to a requirement that $R_{in} \cdot \Delta \geq N$. On the other hand, the expiration period should be short enough to avoid keeping too much state at the server, which means $R_s \cdot \Delta \leq M$. These two constraints can serve as guidelines for RainCheck Filter configuration.

Since the server typically has no control over N and may be unable to immediately increase the downlink capacity or memory, we can set Δ to be $\frac{\tilde{N}}{R_{in}}$, where $\tilde{N}$ is the estimate of the number of clients. The server can obtain $\tilde{N}$ based on the recent history, and adjust Δ accordingly if the estimation changes. If the Δ value does not satisfy the second constraint, the server can either reduce the request processing rate to $R_s = \frac{M}{\Delta}$ or increase its memory size if possible. Note the reducing R_s to $\frac{M}{\Delta}$ does not affect the MWT guarantees since R_s is still higher than $\frac{L}{\Delta}$, which is required for obtaining the MWT bound in the worst-case scenario in Section 4.3. We show how to bound and determine R_s in Section 4.4.4.

4.3.3 Security Benefits

We design RainCheck Filter such that it does not expand the attack surface: rainchecks are protected against forgery with Message Authentication Codes that can be generated and validated at line rate. RainCheck Filter prevents traffic amplification attacks since rainchecks are smaller than typical HTTP request/response headers, which are 700-800 bytes [6]. Also, RainCheck Filter is secure against IP spoofing and raincheck misuses (e.g., raincheck reuse, accumulation, sharing, and forgery), thus preventing compromised or greedy clients from gaining an advantage over legitimate clients.

[4]http://www.intel.com/content/www/us/en/enterprise-security/enterprise-security-aes-ni-white-paper.html

4.4 Improvements and Discussion

We demonstrate that RainCheck Filter's MWT guarantees can be further strengthened (i.e., a reduced T) given auxiliary information about clients, and we propose a hybrid solution to improve scalability.

In addition to knowing that the waiting time is bounded, knowing the actual waiting time increases users' patience [124, 24]. To provide users' actual waiting time, we devise a server-side algorithm that accurately estimates each client's waiting time.

4.4.1 Strengthening Waiting Time Guarantees

Distinguishability between bots and legitimate clients. RainCheck Filter treats all clients alike without differentiating bots from legitimate clients. By rejecting requests from bots, the server can shorten the maximum waiting time for legitimate clients.

To identify and block bots, which are automated without human users, the server applies reverse Turing tests, such as CAPTCHAs [161], to detect human presence. Additionally, the server can identify bots based on known blacklists[5]. Although existing bot detection schemes are imperfect [169, 29, 28], RainCheck Filter bounds the waiting time such that the more accurate the bot detection scheme is, the lower RainCheck Filter's waiting time bound.

The non-uniform distribution of bots in the Internet. Recent studies on bot population found a skewed distribution of bots: 20 ASes host 41% of bots [107] and 538 ASes host over 90% of bots [102]. To exploit this fact, the server establishes a baseline of the normal request rate per AS during peacetime, and the baseline is used to allocate resources for each AS, similar to the proportional allocation technique introduced in Section 4.2.5. By isolating requests from different ASes, RainCheck Filter lowers the waiting time bound for clients in a less contaminated AS, as bots in the same AS have to compete among themselves before being able to attack the server. This approach also motivates ISPs to remove bots in their domains, as users have incentives to switch to less-contaminated ISPs.

4.4.2 Extension for Better Scalability

To avoid a sudden increase in the bandwidth loads in the rare case when all clients retry concurrently, it is desirable to distribute the bandwidth loads caused by raincheck renewals. We propose a hybrid scheme that combines RainCheck Filter with a coarse-grained scheduling for balancing load distribution.

[5]Composite blocking list. `http://cbl.abuseat.org/`

Similar to the original RainCheck Filter, this hybrid scheme requires each client to renew its raincheck periodically. The novel improvement is the assignment of the time intervals such that all the requests from the same client always fall in the same time interval, thereby reducing the overhead for duplicate detection and rate limiting.

In particular, the server divides the time into non-overlapping time intervals $\delta_i = [i \cdot w, (i+1) \cdot w)$ for some constant w. A raincheck issued during δ_i is valid only during δ_j where $j \in [i + m_{min}, i + m_{max})$ for some required cooling period m_{min} and expiration period $m_{max} = m_{min} + m_\Delta$, where m_{min}, m_{max}, and m_Δ are positive integers. The integer value j is derived such that

$$\begin{aligned} & i + m_{min} \leq j < i + m_{max}, \\ & j \mod m_\Delta = PRF_k(cid) \mod m_\Delta, \end{aligned}$$

where k is a secret key and PRF is a pseudorandom function. These equations have a unique solution.

This construction ensures that every raincheck-carrying request from the same client always comes back during the same time interval. Also, requests are renewed approximately every m_Δ time intervals. Therefore, the server only needs to keep track of the accepted requests during the current time interval, δ_i, for duplicate detection and rate limiting. This hybrid scheme is easier to implement as the server does not need to maintain a sliding window of Δ as in the original RainCheck Filter.

The server may want to update the secret key to increase randomness and minimize the risk of key exposure. Suppose the server would like to completely switch to a new key k' from the beginning of δ_v. To ensure a smooth transition, from the beginning of $\delta_{v-m_{max}}$ the server chooses to use the old key or the new key on a per-request-basis according to the following criterion: if $j < v$ when computing j using the old key, then use the old key; otherwise use the new key.

This hybrid scheme trades flexibility for scalability. At one extreme where the PRF perfectly distributes the clients among m_Δ intervals, the overhead (e.g., storage or bandwidth consumed by the raincheck renewal process) is reduced by an order of m_Δ. In particular, we obtain a MWT guarantee $\lceil \frac{N}{L} \rceil w$ when $w \geq \frac{L}{R_s}$. We omit this proof since it is similar to that in Section 4.3. At the other extreme where the PRF maps every client to the same time interval, it is degenerated to the original RainCheck Filter with $\Delta = m_{max} \cdot w$, which renders a MWT bound $\lceil \frac{N}{L} \rceil m_{max} w$. Since any practical implementation of PRF should generate reasonably randomized outputs given that the key is kept secret, this hybrid scheme is expected to have much better scalability than the original RainCheck Filter.

The server can adjust m_{max} dynamically to distribute the load among $m_{max} - m_{min}$ intervals such that the server is slightly overloaded during each interval. The m_{min} value should be large enough to ensure that requests can return to the server on time.

We now briefly discuss how to set the parameters for the hybrid scheme described in Section 4.4.2. In the hybrid scheme, the server similarly has to satisfy $R_{in} \cdot m_\Delta \cdot w \geq N$ and $R_s \cdot w \leq M$. Given the introduction of an additional parameter, m_Δ, we can fulfill the second criterion first by setting $w = \frac{M}{R_s}$. We then set $m_\Delta = \frac{\tilde{N}}{R_{in} \cdot w}$.

4.4.3 Waiting Time Estimation

Providing human users with feedback of their expected waiting time can help increase their patience [124, 24]. While the waiting time bound T can serve as a loose estimate of the actual waiting time, we desire a better estimate that incorporates the current client status.

We present a *rank estimation* algorithm that allows the server to estimate any client c's rank (i.e., position in the virtual queue) at time t without keeping per-client state. The algorithm refines the estimate by taking into account the number of clients that did not renew their rainchecks, and the server informs the client the estimated rank by piggybacking it on the raincheck.

Efficient and accurate rank estimation is challenging: counting the number of issued rainchecks with a timestamp value smaller than that of client c is inefficient, as it requires maintaining one counter for each client. Moreover, since a client can have multiple valid rainchecks at hand, the server should prevent counting the same client twice.

To provide efficient and accurate estimation, our rank estimation algorithm extends a *probabilistic counting* algorithm proposed by Flajolet and Martin [64], which estimates the number of distinct items in a set using $O(\log N)$ memory, where N is the number of distinct items. The probabilistic counting algorithm keeps a bit vector (referred to as an FM sketch) that is initialized to zeros, and uses a deterministic function to map an item to the ith bit with probability $1/2^i$. A bit is set to 1 if an item is mapped to that bit. Given an FM sketch V the estimated number of distinct items is $\tilde{n} = 2^{lsb_0(V)}/0.77351$, where $lsb_0(V)$ is the lowest-order 0-bit position of V (zero-based indexing). One can reduce the estimation error by averaging the results of multiple FM sketches with different index functions. The FM algorithm guarantees a bounded error such that $Pr[|\tilde{n} - n| < \epsilon N] > 1 - \delta$ using $O(\frac{\log(2/\delta)}{\epsilon^2})$ sketches.

Problem formulation. Let $U^t_{t-\Delta}(x)$ contain all clients obtaining at least one raincheck with a timestamp $\leq x$ during $[t - \Delta, t)$. Let $n^t_{t-\Delta}(x)$ be the size of $U^t_{t-\Delta}(x)$. Since a client in front of c at time t must be in $U^t_{t-\Delta}(TS(\rho_c))$, $rank(c, t) \leq n^t_{t-\Delta}(TS(\rho_c))$, where ρ_c is client c's raincheck that has the smallest timestamp value among all valid rainchecks at time t and TS stands for timestamp.

Estimating each client's rank separately using FM sketches requires $O(N \log N)$ memory. To reduce the overhead, our rank estimation algorithm is designed to answer queries such as "How many distinct

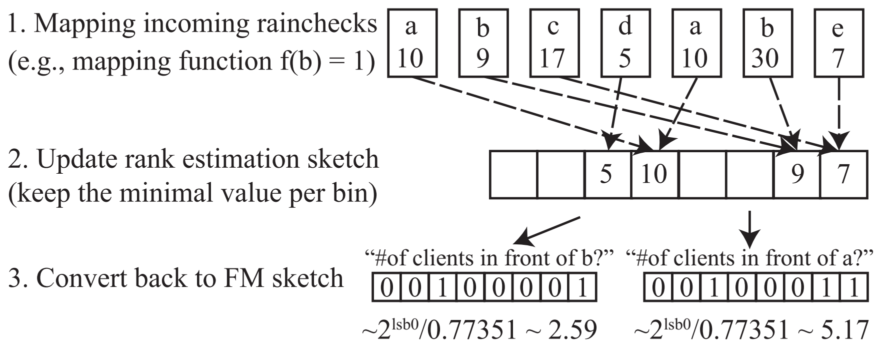

1. Mapping incoming rainchecks
(e.g., mapping function f(b) = 1)
a 10
b 9
c 17
d 5
a 10
b 30
e 7
2. Update rank estimation sketch
(keep the minimal value per bin)
5
10
9
7
3. Convert back to FM sketch
"#of clients in front of b?"
"#of clients in front of a?"
0 0 1 0 0 0 0 1
0 0 1 0 0 0 1 1
~2^lsb0/0.77351 ~ 2.59
~2^lsb0/0.77351 ~ 5.17

Figure 4.3: A rank estimation sketch example.

items have a value lower than x?" for any x, while keeping only $O(\log N)$ number of items.

Figure 4.3 illustrates how the rank estimation sketch works and its relationship with FM sketches. The core idea is as follows. Instead of setting a bit to one, we store the item's value at the mapped position if the value is lower than the currently stored value at that position. To estimate the number of items whose value is lower than x for any x, we convert this rank estimation sketch back to an FM sketch where a 1-bit is set if the corresponding position has a value $< x$. The rank estimation sketch ensures that the resulting FM sketch is the same as the one we can get by running the FM algorithm over items with a value lower than x. In our setting, items are rainchecks, and each raincheck has a value which is a timestamp. Two rainchecks are "identical" if they are from the same client. Hence, using this rank estimation algorithm, we can estimate $n_{t-\Delta}^{t}(x)$ for any x given a time interval $[t-\Delta, t)$.

To estimate a client's rank at time t_{cur} in the virtual queue, the server maintains a rank estimation sketch and resets it periodically every Δ time, such that the sketch accounts for intervals $[(i-1)\Delta, i\Delta)$ for all positive integers i. The rank $rank(c, t_{cur})$ can be approximated by

$$rank(c, t_{cur}) \leq rank(c, i\Delta) \leq n_{(i-1)\Delta}^{i\Delta}(TS(\rho_c)),$$

where $i\Delta \leq t_{cur} < (i+1)\Delta$.

Based on the proof in Section 4.3.1, $rank(c, i\Delta) - rank(c, t_{cur}) \leq (t_{cur} - i\Delta)R_s$. Also, $n_{(i-1)\Delta}^{i\Delta}(TS(\rho_c)) - rank(c, i\Delta) \leq \Delta R_s$. Hence, using $O(\frac{\log(2/\delta)}{\epsilon^2})$ sketches, we ensure that the estimation error is less than $\epsilon N + (t_{cur} - (i-1)\Delta)R_s \leq \epsilon N + 2\Delta R_s$ with a probability higher than $1-\delta$.

4.4.4 Bounding and Determining the Request Processing Rate

To provide MWT guarantees, the bottleneck (e.g., the server) is required to process at least R_s requests per second. We discuss how to ensure a lower bound on R_s under several application scenarios. We also discuss how to estimate R_s when request processing time varies. While accurate estimation of R_s (client's waiting time) can improve user experience, RainCheck Filter's operations and guarantees do not depend on an accurate modeling of R_s.

When the server's access link is the bottleneck, RainCheck Filter has to be installed in front of the access link, such as at a firewall or a load balancer. Since the bandwidth (BW) is the critical resource, raincheck can determine a lower bound of the request processing rate by $R_s \geq \min\{\frac{\text{downlink\_BW}}{\text{max\_req}}, \frac{\text{uplink\_BW}}{\text{max\_resp}}\}$, where max_req and max_resp are the maximum sizes of the request and response packets, respectively. A similar approach can be applied to the case in which an intermediate network link is the bottleneck.

The attacker can also try to exhaust the CPU resource. Although the server has full knowledge about the type of service it offers, the time to complete a request may still be unpredictable beforehand. One possible solution is to consider a raincheck to be an explicit permission for accessing a unit of resources. Moreover, clients can wait longer to obtain higher level rainchecks that give permissions to access more resources. For example, one raincheck can represent 10k CPU cycles or 10ms of the server time. To adopt this modification, either (1) the client divides the task into smaller chunks so that each chunk can be processed in one unit of resources, or (2) the server terminates the process for the accepted request if it has run out of one unit of resources, and returns the necessary information such that the server can resume the process later, if possible. The server being attacked could also decline such computational-expensive requests, which is similar to a safe mode that only supports limited functionality. We leave it as future work to explore such resource allocation policies in the RainCheck Filter framework.

When the request completion time is predictable, the server can allocate resources to multiple request groups, each of which consists of the requests with similar completion time (e.g., based on their service types). Grouping similar requests not only improves the accuracy of waiting time estimation but also allows the server to apply RainCheck Filter to resource-consuming requests only.

RainCheck Filter is not designed to address memory exhaustion attacks such as TCP SYN flooding or slow HTTP attacks. Possible technical defenses to mitigate such attacks include (1) pushing state back to the client, such as SYN cookies and (2) setting an explicit timeout on each buffered request.

4.5 Evaluation

To validate that RainCheck Filter effectively simulates an infinite buffer and thereby enables bounded waiting time with low variance in the presence of flash crowds or DDoS attacks, we evaluate RainCheck Filter using the NS-3 simulator. Our simulation measures the waiting time of legitimate users in two cases: (1) a flash-crowd case where a large number of legitimate and malicious users simultaneously try to access a server within a short-time period, and (2) a DDoS attack case where a server is flooded by bots.

For both cases, we compare results among (a) RainCheck Filter, (b) a traditional client-server model with no protection, and (c) a computational puzzle model.

4.5.1 Flash-Crowd Effect

General setting. We consider a scenario where 100,000 legitimate and malicious clients attempt to connect to a server that can buffer max. 200 requests, and server's request processing time follows an exponential distribution with an average of 5 milliseconds. Every client makes one request where the initial request time is uniformly distributed across a 100-second interval, and the server experiences on average 1,000 incoming requests per second, flooding the server. Next we briefly describe the server and client models to simulate the flash-crowd effect. Note that malicious clients behave exactly like legitimate clients.

RainCheck Filter. We model the server and clients that follow the protocol described in Section 4.2.3 and 4.2.4, respectively.

Unprotected. The unprotected server implements a standard FIFO queue, and informs the client whenever its request is dropped. The client continues to resubmit a request that the server drops until the request is accepted by the server.

Computational puzzles. We model a client that solves a computational puzzle before sending a request [164]. We model the puzzle server with a priority queue which uses the puzzle-level as the priority metric. For requests with the same priority level, the server processes them based on their arrival order.

To send an initial request, the puzzle client solves a level-1 puzzle. Whenever the request with level-n is denied, the client solves a level-$n + 1$ puzzle and resubmits its request. Once the client reaches the maximum puzzle level m, the client continues to solve different level-m puzzles until its request is accepted by the server.

Results. As shown in Figure 4.4, scatter plots for the unprotected and puzzle-based models indicate that clients suffer from high variance for their requests to be accepted. On the other hand, RainCheck Filter supports low variance such that the waiting time steadily increases as more clients send requests. More specifically, the ordering of the requests based on the their generation time is well-preserved with respect to the times that they are served. In fact, the scatter plot for the raincheck server is almost identical to that for an ideal server that has an infinite buffer.

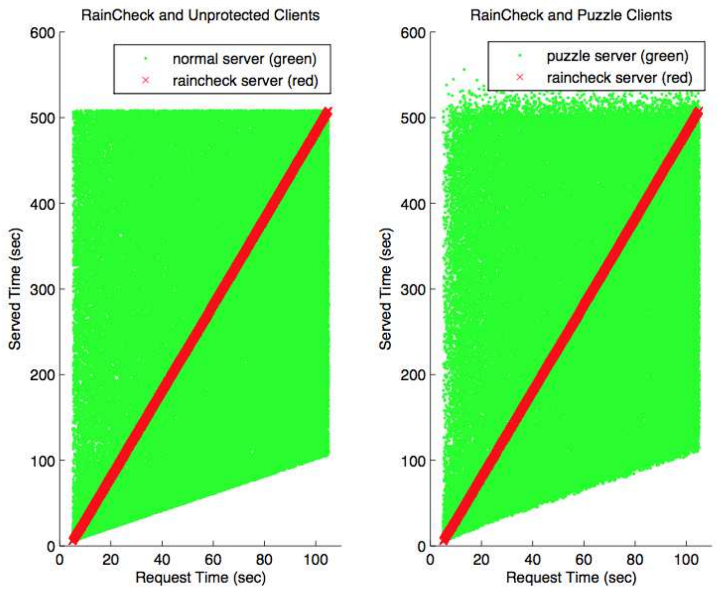


Figure 4.4: Scatter plots of the initial request time vs. served time.

4.5.2 Flooding Attacks

General setting. We consider a scenario where 10,000 legitimate clients attempt to connect to a server that can buffer 200 requests, and server's request processing time follows an exponential distribution with an average of 5 milliseconds. The number of bots varies from 1 to 70,000 to observe the relationship between the amount of attack traffic and the changes in waiting time. Every client makes one request and the initial request time is uniformly distributed across a 100-second interval.

To simulate the DDoS attack case, we model bots that adopt optimal strategies to incapacitate the server. The legitimate client and the server models are as described in Section 4.5.1.

Results. As shown in Figure 4.5, 1,090 bots successfully flood the unprotected server, causing client requests to be dropped and hence resulting in infinite waiting time. For the puzzle-based model, our simulation parameters cause 65,536 bots to jointly compute level-16 puzzles at the rate of 200 puzzles per second, such that the server spends most of its resource in handling the bot's request without accumulating them in its queue. Thus, any request with lower priority in the buffer would experience a long queuing delay.

For RainCheck Filter, we observe a linear increase in the waiting time as the number of bots increase. This is because each client's request is served after serving the requests with earlier timestamp. Since the number of bots' requests that precedes the client's request increases linearly as the number of bots increases, client's waiting time also increases linearly.

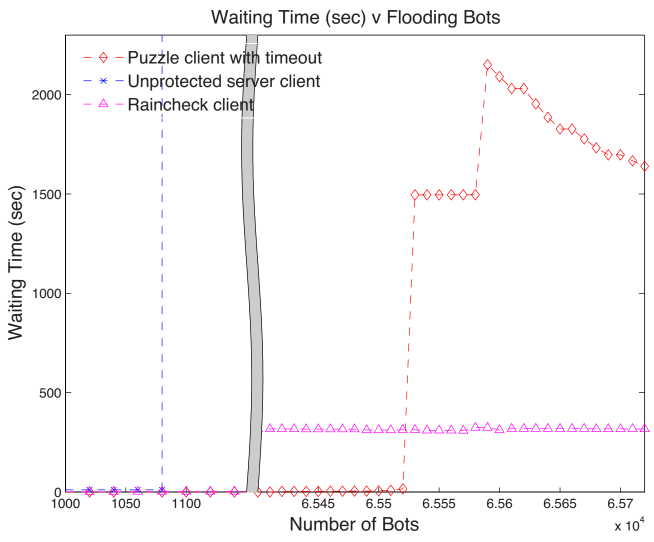


Figure 4.5: Maximum waiting times under flooding attacks.

4.6 Prototype Implementation

To demonstrate the feasibility of RainCheck Filter's incremental deployment, we implement a prototype at the application layer by incorporating RainCheck Filter to HTTP and as a client-side and server-side proxy, therefore leaving the current network protocol stack intact. This prototype bounds the waiting time of legacy client browsers when they access webpages that consume the server's critical resource, e.g., a PHP embedded HTML supporting database queries. The prototype consists of three components: (1) a client running a Firefox web browser, (2) an Apache web server, and (3) a raincheck module (e.g., installed in a load balancer) serving the HTTP protocol. We use tinyproxy[6], an open-source proxy, as the browser's agent to interpret and generate raincheck messages. On the server side, the raincheck module intercepts all traffic to/from the server.

As shown in Figure 4.6, the browser sends a GET request to a critical resource page named `sql.html`. The proxy encodes the raincheck header by attaching "`X-Raincheck-Type:new`" to the request's HTTP header. Once this request arrives at the raincheck module on the load balancer, the module decodes the header and checks whether this request attempts to access the critical resources defined by the back-end server. If so, the raincheck module performs one the following actions:

1. Process the request: If the priority queue is empty, the raincheck module directly forwards the request to the back-end server, and sends back the query result with additional raincheck header ("`X-Raincheck-Type:Served`").

[6]`https://banu.com/tinyproxy/`

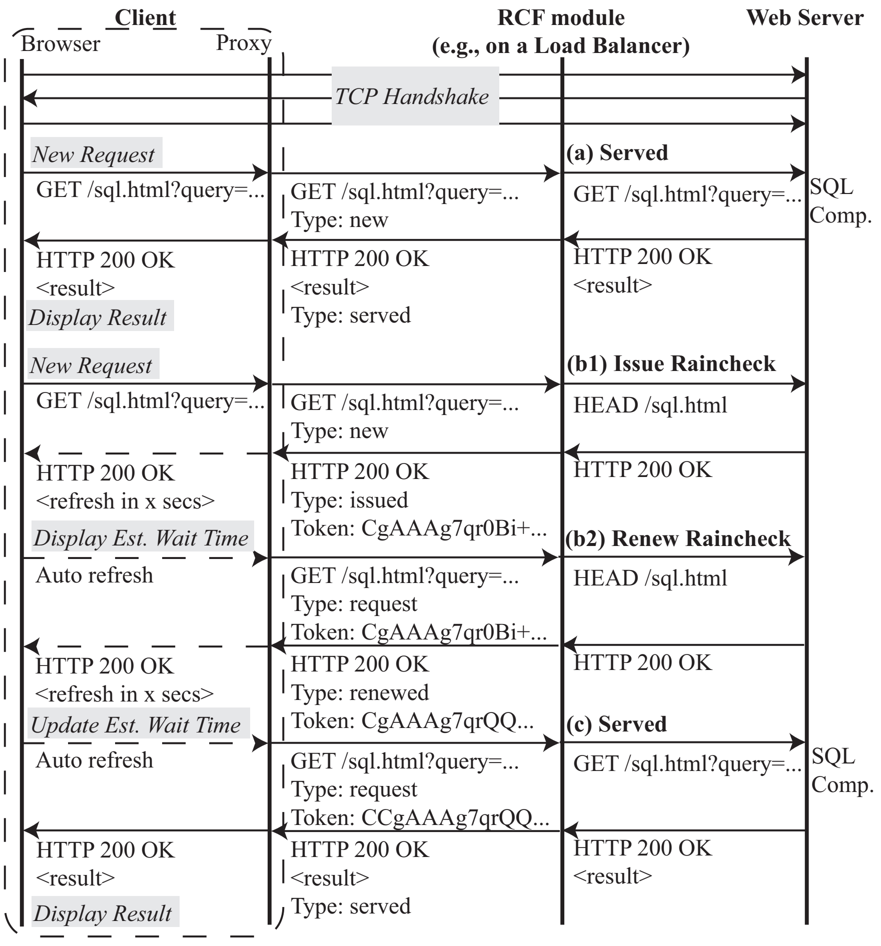


Figure 4.6: RainCheck Filter prototype implementation.

2. Issue a raincheck: RainCheck Filter mangles the original `GET` request to `HEAD` request and delivers the modified request to the server. Once RainCheck Filter receives a response from the server, it embeds a raincheck token by adding "`X-Raincheck-Token:`" string.

The raincheck token is piggybacked to the standard HTTP request/response. For example, we convert `GET` request into `HEAD` request and enforce the server to return the response message with no additional computational cost and minimum bandwidth overhead, e.g., 76 bytes. Therefore, RainCheck Filter can utilize the HTTP header space in this response to "carry" either the issued or renewed raincheck token in a conventional way. This implementation does not require any modifications on the protocol stacks or the operating system such like Linux Kernel. The only required task is to recompute checksums for IP and TCP headers after encoding the HTTP messages.

The raincheck token is encoded in the Base64 format. The proxy parses the raincheck and returns to the browser (1) the estimated waiting time and (2) a webpage containing a meta refresh tag (e.g., `<meta`

`http-equiv="refresh" content="2">`) such that the browser will recontact the server after the specified time. By utilizing the HTTP redirection responses, the proxy delivers the renewed raincheck request (see (b2) in Figure 4.6) and keeps updating the raincheck token until the request is served (see (c) in Figure 4.6).

To evaluate the performance of raincheck implementation, we run an HTTP client written in python to send 1000 `GET` requests sequentially for a static web page under two conditions, with or without the raincheck module. The raincheck module issues one raincheck token and ask the client to use this token to retry. Without the raincheck module, it takes an average of 1.328 milliseconds to get the reply; with the raincheck module, it takes an average of 1.531 milliseconds. This indicates that RainCheck Filter incurs only 0.206 milliseconds additionally per request.

Implementing RainCheck Filter as a TCP option could be a generic defense mechanism that protects all upper layer protocols and applications using TCP. However, raincheck implementation on TCP requires the modification of operating system kernels or installation of privileged modules (e.g., kernel modules), which is intrusive and thus may be less favorable than an incremental deployment that modifies neither the client nor the server. We consider this option as our future work.

4.7 Summary

The technology advances introduced unfortunate side effects: internet users are more impatient than ever, and DDoS attackers practice sophisticated and powerful strategies. To help users control their impatience in the presence of DDoS attacks, we propose RainCheck Filter, a lightweight DDoS mitigation primitive that bounds the waiting time of a legitimate client using a cryptographic token called a raincheck regardless of other clients' strategy. Unlike existing DDoS mitigation schemes that waste users' bandwidth or computational resources to access (possibly flooded) servers, RainCheck Filter requires users to simply wait and revisit the servers before rainchecks expire.

Chapter 5

Large Flow Detection

Chapter 3 focuses on the design of a DDoS-limiting mechanism that isolates attack traffic by means of bandwidth allocation. It assumes that every endpoint AD regulates traffic on a per-flow basis, such that the allocations can be efficiently enforced throughout the network without keeping per-flow state at intermediate ADs.

However, this assumption may be difficult to meet, as in practice an endpoint AD may lack incentives to throttle its customers' traffic. Also, ADs in different ISDs may not trust each other for traffic regulation.

The relaxation of this assumption (i.e., per-flow monitoring) requires an efficient monitoring algorithm that allows individual ADs to accurately detect flows using more than their allocations. Accuracy is important here because failing to regulate malicious flows or falsely catching legitimate flows results in collateral damage in the sense that the bandwidth guarantees of legitimate flows are compromised.

This chapter explores how to enforce bandwidth allocation in an efficient manner while minimizing collateral damage. Specifically, we present an efficient flow-monitoring algorithm that is exact outside an ambiguity region—it robustly catches all large flows and protects all small flows from false accusation. Moreover, unlike prior approaches that characterize flows based on the average rate over a long period of time, we characterize flows based on both the average rate and burstiness, thus enabling the detection of bursty flows that violate the allocation.

Motivation: Challenges of developing an accurate and efficient large-flow detector. Flows that violate their allocations may undermine the guarantees of other legitimate flows. Therefore, it's important to catch malicious flows that use more than their allocations. In the ideal case, we want perfect detection of flows using more than a threshold of bandwidth. However, since keeping per-flow state is too expensive, prior work often faces one or both of the following limitations: (1) They trade accuracy or timeliness for scalability, resulting in false positives (i.e., falsely catching small flows), false negatives (i.e., failing to catch large flows), or delayed detection. (2) They check the average throughput over a long time interval only and thereby cannot catch bursty flows. Moreover, due to the increasing number of network flows and the high cost of fast memory (e.g., SRAM), we need scalable designs that maintain as less memory state in routers as possible (i.e., no per-flow state) and operate at near line-rate.

Our solution: A large-flow detector that monitors over arbitrary windows and is exact outside an ambiguity region. To address these challenges, we propose LFD, a new online deterministic algorithm that presents two novel concepts. First, we consider a relaxed notion of exactness: the algorithm outputs a set of large flow candidates such that every flow whose volume is ever above a high-bandwidth threshold function during some time window is contained in the set, and no flow whose volume is consistently lower than a low-bandwidth threshold function over any time window is contained. The gap between the high-bandwidth threshold and the low-bandwidth threshold, which is referred to as an *ambiguity region* in this chapter, allows us to improve scalability while maintaining exactness outside this region. Second, rather than monitoring the average throughput over a long time, LFD monitors every possible time window (referred to as the *arbitrary time window model*) such that both large and bursty flows can be detected immediately, thus making it harder for the adversary to bypass detection. By contrast, most prior work considers the landmark or sliding window model, in which only a subset of time windows are monitored. These two new properties are obtained at the cost of not being able to estimate flow sizes, which many of the prior works can do. But for our purpose of detecting large flows, a binary detector is sufficient.

Surprisingly, despite LFD's strong guarantees, we show in our analysis that LFD requires extremely small amounts of memory that fit into on-chip SRAM for line-speed packet processing. We discuss implementation details to further demonstrate LFD efficiency. LFD is highly scalable because it focuses on the accurate classification of large and small flows; it does not aim to estimate flow volumes or identify medium flows, which several prior approaches achieve. In addition to our theoretical analysis, we also evaluate LFD using extensive simulations based on real traffic traces. We demonstrate that existing approaches suffer from high error rates in adversarial environments, whereas LFD can effectively detect large flows in the face of both flooding and burst DoS attacks [71, 93].

Contributions. Our main contributions are as follows.

- We propose a deterministic streaming algorithm that robustly catches all large flows and protects all small flows. Two novel settings distinguish LFD from previous work: it monitors flows over arbitrary time windows and considers a relaxed definition of exactness.

- We rigorously prove the two guarantees—catching all large flows and protecting all small flows—without making assumptions about the traffic distribution. In other words, LFD is resilient to a worst-case attacker who knows the algorithm's internal state. By contrast, prior work cannot achieve such guarantees even under a weaker attacker model.
- Our numerical analysis shows that LFD can operate at 40 Gbps high-speed links using only hundreds of bytes of on-chip SRAM, which is substantially smaller than the memory consumption of many existing schemes. We also provide guidelines on how to configure LFD to satisfy application-specific requirements.
- We compare LFD with two closely related proposals [58, 57] via comparative analysis and extensive simulations based on real and synthetic traffic traces. The results confirm that LFD consistently catches all large flows without misclassifying small flows, whereas prior approaches result in high false detection rates, especially under flooding attacks.

5.1 Problem Definition

Our goal is to design an efficient arbitrary-window-based algorithm which is exact outside an ambiguity region. In this section, we present the system model, formulate the large flow problem over arbitrary windows, and summarize our design goals.

5.1.1 System Model

Flow identifiers. Generally, packets are classified into flows based on the *flow identifiers* (or flow IDs) derived from the packet header fields.[1] Because our approach to large-flow detection is generic, we make no assumption on the definition of flow IDs. As in prior traffic monitoring work, we assume flow IDs are unforgeable, which can be achieved using existing mechanisms such as ingress filtering [62], and source authentication techniques such as accountable IPs [10], Passport [109], and ICING [126].

Packet streams. Let $\mathcal{X}$ be the packet space. We consider a packet stream $X = \langle x_1, \cdots, x_k \rangle$ coming through a link of capacity ρ, where $x_i \in \mathcal{X}\ \forall i = 1 \cdots k$. Packets in X are processed in sequence by a detection algorithm for identifying large flows. The algorithm can only make one pass over the packet stream due to the high link capacity and limited memory.

[1]While the flow definitions vary depending on the application, in most applications a flow consists of packets that share one or more header fields, such as the source IP, destination IP, source port, destination port, and the protocol number.

For a packet x, we denote by $\mathsf{time}(x)$ the time when the packet is received, by $\mathsf{size}(x)$ the size of the packet, and by $\mathsf{fid}(x)$ the flow ID of the packet. The traffic volume of a flow f during a time window $[t_1, t_2)$ is defined as $\mathsf{vol}(f, t_1, t_2) \triangleq \sum_{x \in \mathcal{X}, \mathsf{fid}(x)=f, t_1 \leq \mathsf{time}(x) < t_2} \mathsf{size}(x)$.

Synopses. A traffic synopsis is a data structure that summarizes flows and can be used to answer queries regarding certain flow statistics. When a new packet x arrives, the algorithm updates its traffic synopsis based on x's flow ID, size, and arrival time. Formally, a large-flow detection algorithm supports three operations over a synopsis S:

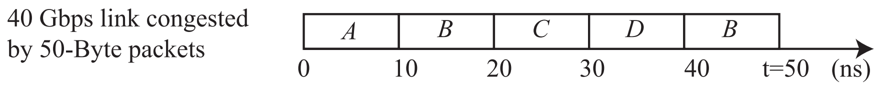
- $\mathsf{Init}(params) \to S_0$. The initialization operation takes as inputs the large-flow definitions, desired detection accuracy, etc.
- $\mathsf{Update}(S_{i-1}, x_i) \to S_i$. The update operation outputs an updated synopsis S_i by incorporating the new packet x_i into the previous synopsis S_{i-1}. For convenience, we denote $\mathsf{Update}(S_i, \langle x_{i+1}, \cdots, x_{i+j} \rangle) = \mathsf{Update}(S_{i+1}, \langle x_{i+2}, \cdots, x_{i+j} \rangle) = S_{i+j}$.
- $\mathsf{Detect}(S_i, x_i) \to b \in \{0, 1\}$. The detection operation evaluates S_i to determine if x_i belongs to a large flow.

Time window models. Prior approaches to large-flow detection can be classified into three main categories, based on the type of time window they monitor: landmark window [117, 44, 83, 114, 116, 58, 61, 38], sliding window [68, 13, 99], and arbitrary window [57].

In the landmark window model, each time window starts at the closest landmark in the past (e.g., a landmark is placed every five seconds) and ends at the current time. In the sliding window model, recent traffic is considered more important than old traffic, so the time window begins at some recent time in the past. The window slides as new packets arrive, such that the measurement incorporates the new packets and excludes the oldest packets. Finally, the arbitrary window model monitors every time window ending at the current time. It is more difficult to evade detection in this model than in the others, as illustrated in Figure 5.1. Note that while the arbitrary window model covers every possible window, flows can still evade detection if the algorithm is inaccurate.

5.1.2 Large-Flow Problem

Small, medium, and large flows. A flow f is a *large flow* (over arbitrary windows) if there exists a time window $[t_1, t_2)$ over which its volume $\mathsf{vol}(f, t_1, t_2)$ exceeds a high-bandwidth threshold function $\mathsf{TH_h}(t_2 - t_1)$. A flow is a *small flow* if its volume $\mathsf{vol}(f, t_1, t_2)$ is lower than a low-bandwidth threshold



Landmark window model (Landmark at 0)	Examine flows in [0, t) → flow B evades detection
Sliding window model (window size=30ns)	Examine flows in [t-30, t) → flow B evades detection
Arbitrary window model	Examine flows in [s, t) for all t > s ≥ 0 → flow *B* is a large flow over [10, 50) and can be detected

Figure 5.1: In this example, a flow is large if it sends more than $40Mbps \cdot w + 500kb$ for any time window of size w. Although flow B violates the limit over the time window $[10, 50)$, it can only be caught in the arbitrary window model.

function $\mathsf{TH}_\ell(t_2 - t_1)$ over all possible time windows $[t_1, t_2)$. The rest are defined as medium flows, i.e., flows in an ambiguity region.

In this chapter, we define the two threshold functions in the form of leaky bucket descriptors: $\mathsf{TH}_\mathsf{h}(t) = \gamma_h t + \beta_h$ and $\mathsf{TH}_\ell(t) = \gamma_\ell t + \beta_\ell$, where $\gamma_h > \gamma_\ell > 0$ and $\beta_h > \beta_\ell > 0$.[2] Although selecting appropriate parameters largely depends on the targeted application, we provide guidelines for selecting these parameters in Section 5.3.

Exact-outside-ambiguity-region large-flow problem. As exact solutions are inefficient, we consider a relaxed notion of exactness:

Definition 6. *Given a packet stream, the exact-outside-ambiguity-region large-flow problem over arbitrary windows returns a set of flows $\mathcal{F}$ such that (1) $\mathcal{F}$ contains every large flow, and (2) $\mathcal{F}$ does not contain any small flow.*

A *positive* is when a flow is added to $\mathcal{F}$, and a *negative* is when a flow is not added to $\mathcal{F}$. Hence, a False Positive of small flow (FP_s) occurs when the detection algorithm wrongly adds a small flow, and a False Negative of large flow (FN_ℓ) occurs when it fails to include a large flow.

5.1.3 Adversary Model

We consider a practical setting where the senders can be malicious. Particularly, we consider three types of typical attacks: evasion attacks, framing attacks, and denial-of-service (DoS) attacks. In an evasion attack, the attacker attempts to cover the trace of a large flow to prevent it from being detected. In a framing

[2]Prior work in the landmark window model often defines the high-bandwidth threshold to be a fraction of the link bandwidth, e.g., $\gamma_h = 0.01$ and $\beta_h = 0$. However, because every flow will violate the threshold over a sufficiently small time window (e.g., windows containing only one packet), it is infeasible to adopt this fraction-based definition when it comes to the arbitrary window model.

attack, the attacker attempts to incriminate a small flow to make it look as if it is misbehaving. In a DoS attack, the attacker attempts to take the detector offline by depleting its scarce memory or computational resources or by exploiting the worst-case behavior of the detection algorithm.

5.1.4 Design Goals

We aim to design an efficient algorithm which is exact outside an ambiguity region to solve the large-flow problem (defined in Section 5.1.2). Our main goals are as follows:

Exactness outside an ambiguity region. To achieve exactness outside an ambiguity region in traffic monitoring, we desire a deterministic monitor algorithm which identifies every large flow including bursty flow (i.e., no FN_ℓ), and protects every small flow (i.e., no FP_s) with no assumption on the input traffic or attack pattern. Hence, the high-bandwidth and low-bandwidth thresholds are also called the no-FN_ℓ and no-FP_s thresholds in this chapter, respectively.

Scalability. Although using per-flow leaky buckets enables the exact and instantaneous detection of large flows, keeping per-flow state is impractical due to the large number of flows in the Internet. Hence, the algorithm should require few per-packet operations and maintain a small router state that fits in fast yet scarce storage devices (e.g., on-chip SRAM or even registers) regardless of input traffic or attack pattern, such that the detection algorithm can operate at line rate.

Fast detection. The algorithm should promptly catch large flows to minimize collateral damage. Particularly, for a large flow violating the high-bandwidth threshold over $[t_1, t_2)$, the algorithm should detect the flow no later than $t_2 + t_{process}$, where $t_{process}$ is the time it takes to process a packet.

5.2 Algorithm

In this section, we first investigate and prove the *no-FP_s* and *no-FN_ℓ relationships* between landmark and arbitrary window models, These relationships are useful for constructing large-flow algorithms over arbitrary windows. Based on these relationships, we develop LFD, a streaming algorithm that efficiently solves the large-flow problem over arbitrary windows (as defined in Definition 6) with exactness outside an ambiguity region. Finally, we discuss the implementation and optimization techniques in detail and numerically demonstrate that LFD can operate at high-speed links while using only hundreds bytes of on-chip SRAM.

It is important to investigate the relationships between landmark and arbitrary window models, as it enables us to draw on the rich experience of research on the large-flow problem over landmark windows [117, 44, 83, 114, 116, 58, 61, 38] for designing arbitrary-window algorithms. Particularly, we are interested in knowing whether and to what extent we can leverage existing landmark-window algorithms to build arbitrary-window ones. The technical contributions of this chapter include proving two theorems that shed light toward a systematic approach applying existing landmark-window algorithms to arbitrary-window algorithms.

The design of LFD is guided by the no-FN$_\ell$ theorem (i.e., Theorem 8). LFD leverages the Misra-Gries (MG) algorithm [117], which finds all frequent items in a data stream in one pass but may falsely include non-frequent items. The MG algorithm works over landmark windows in the sense that the landmark is at the beginning of the data stream. The research challenges here include (1) how to preserve MG's no-FN$_\ell$ property (over landmark windows) when porting it to the arbitrary window model, and (2) how to achieve the no-FP$_s$ property when processing packets in one pass. LFD modifies the MG algorithm in several novel ways to effectively address the above challenges. Interestingly, despite these simple modifications, we prove that LFD achieves both no-FP$_s$ and no-FN$_\ell$ properties over arbitrary windows, thereby providing strong guarantees regardless of input traffic.

5.2.1 Relationships Between Landmark and Arbitrary Windows

Here is a straightforward yet inefficient solution to the exact-outside-ambiguity-region large-flow problem over arbitrary windows: the algorithm divides the problem into multiple sub-problems that each can be handled by a landmark-window algorithm, $\mathcal{L}$. More concretely, let L^i be a copy of $\mathcal{L}$ that monitors a time window starting from packet x_i and ending at the current time. For every newly arrived packet x_i, the algorithm initiates L^i, and adds x_i to the new as well as all previous copies, $L^1, L^2, \cdots, L^i$. Then the algorithm combines the answers returned by $L^1, L^2, \cdots, L^i$. This naive solution is correct, but requires space linear in the length of the traffic stream, which is prohibitively expensive.

To make it more efficient, the key idea is to eliminate redundant copies of $\mathcal{L}$. To show why this is possible, we formally state two relationships between landmark windows and arbitrary windows.

No-FP$_s$ relationship. We observe that only one copy of $\mathcal{L}$ is needed to achieve the no-FP$_s$ property over arbitrary window. Specifically, Theorem 7 states that if an algorithm ensures (L1) no FP$_s$ in the landmark window model, then it also ensures (A1) no FP$_s$ in the arbitrary window model.

Theorem 7. *If an algorithm satisfies*

- L1: *For all t, it never reports a flow whose volume is below $\gamma'_\ell t + \beta'_\ell$ over time interval $[0, t)$.*

then it must also satisfy

- A1: *It never reports a flow whose volume is below $\gamma_\ell(t_2 - t_1) + \beta_\ell$ over time interval $[t_1, t_2)$ for all $t_2 > t_1$.*

when $\gamma'_\ell = \gamma_\ell$ and $\beta'_\ell = \beta_\ell$.

Proof sketch: L1 implies A1 because if a flow sends less than $\gamma_\ell(t_2 - t_1) + \beta_\ell$ for all intervals $[t_1, t_2)$, it must also send less than $\gamma'_\ell t + \beta'_\ell$ for all intervals $[0, t)$ when $\gamma'_\ell = \gamma_\ell$ and $\beta'_\ell = \beta_\ell$. □

No-FN$_\ell$ relationship. The no-FN$_\ell$ relationship is more challenging to prove. We observe that only one copy of $\mathcal{L}$ is needed to achieve the no-FN$_\ell$ property as well if *$\mathcal{L}$'s traffic synopsis is "similar" to the initial state* throughout the execution of the algorithm, as checking such a synopsis is roughly equivalent to checking all of $L^1, L^2, \cdots, L^i$. In other words, we can keep only one synopsis which somehow approximates the synopsis in each sub-problem.

Formally, we define a distance metric $\mathsf{dis}(S, S')$ quantifying the similarity of two synopses:

$$\mathsf{dis}(S, S') \triangleq \min_{X, S' = \mathsf{Update}(S, X)} \mathsf{tspan}(X),$$

where the time span of a packet sequence is defined as $\mathsf{tspan}(X) = \max_{x \in X} \mathsf{time}(x) - \min_{x \in X} \mathsf{time}(x)$.

Theorem 8 states if an algorithm ensures (L2) no FN$_\ell$ in the landmark window model and (L3) its synopsis is bounded, it also ensures (A2) no FN$_\ell$ in the arbitrary window model.

Theorem 8. *If an algorithm satisfies*

- L2: *For all t, it reports all flows whose volume exceed $\gamma'_h t + \beta'_h$ over time interval $[0, t)$.*
- L3: *Throughout the execution of $\mathcal{L}$, $\mathsf{dis}(S_0, S_i) \leq \Delta$, where Δ is a small constant and $S_i = \mathsf{Update}(S_0, \langle x_1, \cdots, x_i \rangle)$.*

then it must also satisfy

- A2: *It always reports a flow whose volume exceeds $\gamma_h(t_2 - t_1) + \beta_h$ over time interval $[t_1, t_2)$ for some $t_2 > t_1$.*

when $\gamma_h(t_2 - t_1) + \beta_h \geq \gamma'_h(t_2 - t_1 + \Delta) + \beta'_h$.

Proof sketch: Let $\mathcal{L}$ be an algorithm satisfying L2 and L3. Let f be a flow that sends more than $\gamma_h(t_2 - t_1) + \beta_h$ over some time interval $[t_1, t_2)$, and t_2 is the smallest among all possible values if the flow f violates the spec multiple times. To prove this No-FN$_\ell$ relationship, in the following we show that $\mathcal{L}$ can catch any f when $\gamma_h = \gamma'_h$ and $\beta_h \geq \beta'_h + \gamma_h \Delta$, thus satisfying A2 as well.

For convenience, we denote by X_a^b the incoming packet stream between time interval $[a, b)$. Since $\mathcal{L}$ satisfies L3, the synopsis state of $\mathcal{L}$ is always bounded, so $\mathsf{dis}(S_0, S^{t_1}) \leq \Delta$ where S^{t_1} is its synopsis state at t_1.

Based on the definition of the distance function, there exists a packet sequence X' with a time span less than Δ and $\mathsf{Update}(S_0, X') = S^{t_1}$. In other words, from the algorithm's perspective, $\mathsf{Update}(S_0, X_0^{t_2}) = \mathsf{Update}(S_0, \mathcal{X}' \| X_{t_1}^{t_2})$, i.e., the two packet sequences produce identical synopses. As a result, if the algorithm $\mathcal{L}$ can detect f in $X' \| X_{t_1}^{t_2}$ then it can also detect f in $X_0^{t_2}$ because the output of the detection function, Detect, solely depends on the synopsis.

Moreover, by construction, f sends more than $\gamma_h(t_2 - t_1) + \beta_h$ in the new sequence $X' \| X_{t_1}^{t_2}$, whose time span is $t_2 - t_1 + \Delta$. Therefore, $\mathcal{L}$ can detect f in the new sequence because $\gamma_h(t_2 - t_1) + \beta_h \geq \gamma'_h(t_2 - t_1 + \Delta) + \beta'_h$ holds when $\gamma_h = \gamma'_h$ and $\beta_h \geq \beta'_h + \gamma_h \Delta$. Hence, $\mathcal{L}$ can also detect f in the original stream, $X_0^{t_2}$. □

We note that L1 and A2 contradict each other for any parameter selection: For any $\gamma'_\ell, \beta'_\ell, \gamma_h$, and β_h, consider an interval $[t_1, t_2)$ satisfying $t_1 = t_2 - \epsilon$ and $t_2 > \frac{\beta_h - \beta'_\ell + \gamma_h \epsilon + 1}{\gamma'_\ell}$. Then a flow sending $\gamma_h(t_2 - t_1) + \beta_h + 1$ over $[t_1, t_2)$ will violate the high bandwidth threshold over $[t_1, t_2)$ but comply with $\gamma'_\ell t_2 + \beta'_\ell$ over $[0, t_2)$. That is, no algorithm can satisfy (A2, L2, L3) and (A1, L1) at the same time.

The above two theorems can be viewed as guidelines for designing new arbitrary-window algorithms based on existing landmark-window algorithms.

5.2.2 Algorithm Construction

Several existing landmark-window approaches [117, 44, 83, 114, 116, 58, 38] satisfy L2 when β'_h is set to zero. Among these approaches, we observe the MG algorithm can be made to satisfy L2 in a general setting (i.e., β'_h can be non-zero) as well as L3 with slight modifications. As a result, we choose to leverage the MG algorithm for designing LFD.

We prove in the next section that LFD's design ensures L2 and L3, and therefore achieves the no-FN$_\ell$ property (i.e., catching every large flow) based on Theorem 8. We also prove that LFD achieves the no-FP$_s$ property (i.e., protecting every small flow), whereas the MG algorithm requires a second pass to remove false positives in the landmark window model.

Background of the MG algorithm. We briefly review the MG algorithm, which inspires our design. The MG algorithm [117] finds the exact set of frequent items (defined as items that appear in a stream of m items for more than $\frac{m}{n+1}$ times) in two passes with only n counters. This algorithm generalizes the Majority

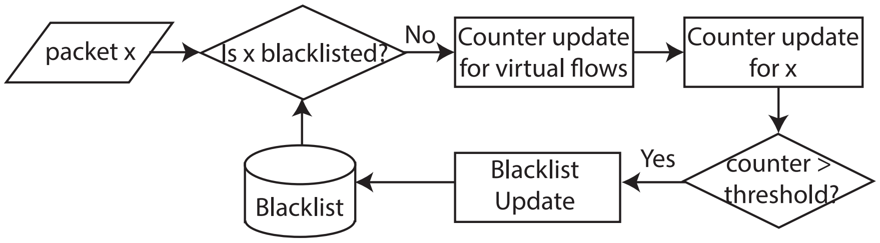


Figure 5.2: LFD's decision diagram.

algorithm [22, 63], which focuses on the case when $n = 2$. The same generalization was rediscovered by Demaine et al. [44] and Karp et al. [83].

The MG algorithm assumes an associative array of counters indexed by items. Counters are initialized to zeros. We say an item is stored if its counter is above zero. For each incoming item e, the MG algorithm works as follows. (1) If e is stored (i.e., $ctr[e] > 0$), then increase $ctr[e]$ by 1. (2) Else if the number of non-zero counters is less than n, $ctr[e] = 1$. (3) Otherwise, decrease all non-zero counters by 1.

Since there are at most n non-zero counters kept at any time, the storage overhead is $O(n)$. This can be easily extended to items with positive weights. After the first pass, the MG algorithm guarantees that every frequent item has a non-zero count, and a second pass is required to remove falsely included infrequent items.

The correctness of this algorithm can be shown intuitively: suppose an item e appears more than $\frac{m}{n+1}$ times, but not remains at the end. The total count would have been reduced by more than $\frac{m}{n+1} \cdot (n+1) = m$ counts during the execution, which is impossible since it is more than the total number of items.

LFD Overview. Figure 5.2 illustrates LFD's decision diagram for each incoming packet. At a high level, LFD works similarly to the MG algorithm except three crucial distinctions:

- **Blacklist:** LFD keeps a blacklist $\mathcal{F}$ that stores identified large flows. Counters are updated only if the flow ID of the packet is not blacklisted. The main purpose of keeping the blacklist is to avoid increasing a flow's counter when the counter value has already exceeded a *counter threshold*, β_{TH}. Additionally, we can avoid spending unnecessary resources on accounting blacklisted flows. We discuss how to limit the blacklist's size in Section 5.2.3.
- **Counter threshold:** A flow is added to the blacklist if its associated counter value exceeds a threshold β_{TH}. Setting a counter threshold together with blacklisting ensure counter values are always confined, i.e., $\leq \beta_{TH} + \alpha$, where α is the maximum packet size.
- **Virtual traffic:** In contrast to the frequent-item problem, the large-flow problem has to take the idle time between two consecutive packets into account so as to accurately detect large flows with respect

to the link capacity. LFD handles this by virtually filling the unused bandwidth with virtual traffic. Virtual traffic consists of multiple virtual flows, each of which complies with the low-bandwidth threshold to avoid unnecessary alarms.

Algorithm description. Algorithm 1 describes how LFD works. As the MG algorithm, LFD keeps n counters, each initialized to zero. Counters are stored in an associative array indexed by flow IDs, and the number of non-zero counters never exceeds n. Since a packet of size w can be viewed as w uni-sized items, LFD counters are increased and decreased by the size of the packets. We denote by C the set of non-zero counters.

Algorithm 1 LFD

```
 1: Initialization ($S \leftarrow \mathsf{Init}(n)$, Line 8-9)
 2: for each packet $x$ in the stream do
 3:     if $x$'s FID $f$ is not blacklisted ($f \notin \mathcal{F}$) then
 4:         Update counters for virtual traffic (Line 18-22)
 5:         Update counters for $x$ ($S \leftarrow \mathsf{Update}(S, x)$, Line 10-17)
 6:         if detect violation ($\mathsf{Detect}(S, x) == 1$, Line 21-22) then
 7:             Add $f$ to blacklist ($\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$)

 8: Initialization, $\mathsf{Init}(n)$
 9: initialize all counters to zeros, $\mathcal{F} \leftarrow \emptyset$, $C \leftarrow \emptyset$

10: Update counters for packet $x$, $\mathsf{Update}(S, x)$
11: if $x$'s FID $f$ is kept ($f \in C$) then
12:     Update $f$'s counter by the packet size $w$ ($c_f \leftarrow c_f + w$)
13: else if less than $n$ counters are kept ($|C| < n$) then
14:     Set $f$'s counter to $w$ ($c_f \leftarrow w$, $C \leftarrow C \cup \{f\}$)
15: else
16:     Decrease all counters by $d = \min\{w, \min_{j \in C} c_j\}$
17:     Set $c_f$ to $w - d$, and $\forall j$ remove $j$ from $C$ if $c_j = 0$

18: Update counters for virtual traffic between $x_i$ and $x_{i-1}$
19: Compute the virtual traffic size, $v$ ($v = \rho t_{idle} - \mathsf{size}(x_{i-1})$, and $t_{idle} = \mathsf{time}(x_i)\mathsf{time}(x_{i-1})$)
20: for each unit $u$ in the virtual traffic, update counters as if $u$ belongs to a new flow (e.g., unit is 1 byte)

21: Detect violation, $\mathsf{Detect}(S, x)$
22: Return whether $x$'s flow counter exceeds threshold ($c_f > \beta_{TH}$)
```

Figure 5.3 gives an example showing how to update counters when $n = 3$, $\beta_{TH} = 10$ and $\alpha = 3$, where α is the maximum packet size. First, since there is an empty counter, flow g is added and its counter value becomes 2, the size of the new packet. Then, since flow b is stored already, its counter is increased by 3. Since the new value exceeds β_{TH}, flow b is blacklisted. The next flow, e, is not stored yet and there is no empty counter, so all counters are decreased by the packet size. Finally, the virtual traffic is divided into single-unit packets with new flow IDs, resulting in the final state.[3]

[3]Conceptually, the counter values are updated as follows: $[3,9,0] \rightarrow [3,9,1] \rightarrow [2,8,0] \rightarrow [2,8,1] \rightarrow [1,7,0] \rightarrow [1,7,1] \rightarrow [0,6,0]$. Section 5.2.3 discusses techniques to accelerate this process.

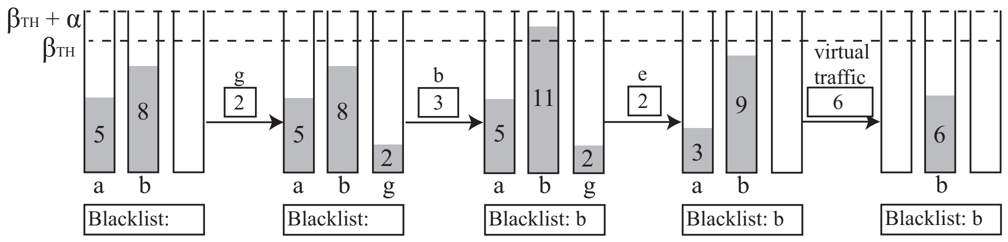


Figure 5.3: Example of LFD's counter update.

Despite LFD's simple operations, work remains to prove the no-FP$_s$ and no-FN$_\ell$ properties and to devise practical parameters. We answer these in Section 5.3.

5.2.3 Data Structure and Optimization

While LFD requires very little memory state, its processing delay may be high in a naive implementation where LFD accesses every counter for each decrement operation (i.e., Line 16 in Algorithm 1). We now present several optimization techniques to reduce the number of memory accesses and the processing time.

Reducing number of memory accesses. To minimize the number of memory accesses per packet, we keep counters in a data structure that allows insertion, deletion, and finding the minimum in logarithmic time. Data structures such as balanced search trees and heaps can satisfy our requirements. Moreover, counter values are not absolute but relative to a *floating ground*, c_{ground}. Hence, the decrement operation, which requires decreasing all counters previously, can now be achieved by elevating the floating ground. The detection function becomes $c_f - c_{ground} > \beta_{BF}$.

The increment operation on Line 12 takes $O(1)$ time using the associative array. Adding a value to an empty counter as described on Line 14 takes $O(\log n)$ time because we have to insert the counter to the data structure. To prevent counter overflow, LFD periodically resets the floating ground to zero and deducts all counters accordingly.

Efficient counter update for virtual traffic. Virtual traffic ensures accurate accounting of unused bandwidth, but efficient implementation is needed to handle virtual traffic at line speed. To enable efficient update, LFD's counters for virtual traffic. We divide virtual traffic into multiple virtual flows in a way to minimize the time to process such virtual flows. The only constraint is that each virtual flow should comply with the low-bandwidth threshold to avoid triggering false alarms.

As Line 20 of Algorithm 1 shows, for each unit u of the virtual traffic, LFD updates its counters (i.e., $\mathsf{Update}(S, u)$) as if the traffic unit belongs to a new flow. We can minimize the number of updates by maximizing the unit size. To avoid false detection, the maximum size per unit is β_{TH} bytes. As β_{TH} must be larger than the minimum packet size (i.e., 40 bytes) for practical use, the overhead of using β_{TH}-byte virtual flows is bounded by the worst-case scenario where the link is congested by minimum-sized packets.

We can further optimize this task based on the following observation: once all counters become empty, they should stay empty until the next real packet comes. Furthermore, since the maximum counter value is $\beta_{BF} + \alpha$, counters will all be empty if the size of the virtual flow $\geq (\beta_{BF} + \alpha) \cdot n$. (A tighter condition is if the virtual traffic size $\geq (\max_j c_j) * n - \sum_j c_j$, but this requires keeping track of the sum of all counters.) In other words, LFD can simply reset all counters to zeros and avoid any update if the virtual traffic size exceeds a certain threshold.

Counter implementation. For efficiency, counters are implemented as integers (e.g., in bytes) rather than non-integer numbers. While packet sizes are always multiples of bytes, the size of virtual traffic may be non-integer, which introduces biases on LFD's guarantees. For example, given an 800Mbps link and a nanosecond time precision at the router, the size of a 1-ns virtual traffic is 0.1 Byte.

We bound such biases with a slightly modified algorithm that adjusts virtual traffic. Let us denote by $\{v_1, v_2, \cdots\}$ the sizes of a sequence of virtual traffic and by $\{v'_1, v'_2, \cdots\}$ the adjusted sizes. We maintain an extra field called "carryover", co, which keeps the amount of uncounted virtual traffic. co is initialized to zero and $-0.5 \leq co < 0.5$ for all time. Virtual flows are adjusted such that $v'_i \leftarrow [v_i + co_i]$ and $co_{i+1} \leftarrow co_i + v_i - v'_i$ where co_i is the value of co before proceeding v_i. By construction, v'_is are all integers, and for any a, b, $|\sum_a^b v_i - \sum_a^b v'_i| = |co_{b+1} - co_a| \leq 1$. In other words, the adjusted virtual traffic differs from the original one by at most 1 unit for any time interval. Consequently, the modified algorithm guarantees to catch flows violating $\mathsf{TH_h}(t) = \gamma_h t + (\beta_h + 1)$ and guarantees not to catch any flow conforming to $\mathsf{TH}_\ell(t) = \gamma_\ell t + (\beta_h - 1)$.

Bounding the blacklist. LFD keeps in memory not only counters but also a blacklist storing detected large flows. While the number of counters is a constant, the blacklist's size, $|\mathcal{F}|$, may grow indefinitely over time.

We propose a simple mechanism to bound $|\mathcal{F}|$, thus preventing algorithmic complexity attacks to overflow the blacklist as follows. The detector reports the blacklist to an administrator when $|\mathcal{F}|$ reaches a pre-defined limit, e.g., $2n$, and then removes currently unmonitored flows (i.e., flows with a zero counter value) from the blacklist. The key observation here is that removing currently unmonitored flows will not affect LFD's no-FN$_\ell$ and no-FP$_s$ guarantees, as in LFD whether a flow will be caught or not does not depend on other flows' behavior. In other words, the administrator keeps a complete list of detected large flows, while the detector maintains a small blacklist that helps avoid increasing a flow's counter when the counter value has already exceeded β_{TH}. The only tradeoff of this mechanism is that LFD may spend unnecessary resources on accounting flows that have been identified as large flows.

Parallelizing LFD. A common way to reduce processing time is via parallelization. LFD can be parallelized at both the algorithm and instruction levels. At the algorithm level, we can randomly distribute the flows (thus the workload) among multiple copies of LFD. At the instruction level, we can access and update multiple counters in parallel using multi-port SRAM[4] when the operations are order insensitive.

5.2.4 Storage and Computational Complexity

Given the above optimization techniques, we analyze LFD's storage and computational overhead.

To operate at line rates on OC-768 (40 Gbps) high-speed links, a typical 3.2 GHz processor has to process 40 million medium-sized (1000 bits) packets per second, which means the per-packet processing time should be at most 32 ns or 76 CPU cycles. In this analysis, we consider the following memory model for commodity routers: CPU has 32 KB L1 cache, 256 KB L2 cache, 20 MB L3 cache, and gigabytes main DRAM memory. Accessing L1, L2, and L3 caches takes 4, 12, and 30 CPU cycles, respectively; accessing the main memory is as slow as 300 cycles.

Storage complexity. LFD keeps an extremely small traffic synopsis and a blacklist of detected flows. The synopsis consists of n counters and a constant number of additional variables for optimization such as the floating ground. In most applications the synopsis will be small enough to fit entirely in the router's L1 cache. For instance, using 100 32-bit counters requires only 400 bytes, which occupy only a negligible portion of the L1 cache. Moreover, we can flexibly tune the counter size to further reduce the memory requirement at the cost of a wider ambiguity region between the no-FP$_s$ and no-FN$_\ell$ thresholds.

Computational complexity. For each packet, LFD looks up and updates one or more counters, and adjusts the internal data structure (e.g., heap) of counters. In LFD, locating and updating a counter requires one memory access in an associative memory. Adjusting the data structure of n counters requires $O(\log n)$ memory accesses.

[4]Special-purpose SRAMs (e.g., multi-port SRAMs) can support multiple read/write simultaneously [174].

Since LFD's state is small enough to fit into the L1 cache as we discussed, the per-packet processing time can be as low as tens of nanoseconds, which is suitable for processing packets at 40Gbps high-speed links.

5.3 Analysis

In this section, we prove the no-FP$_s$ and no-FN$_\ell$ properties. Furthermore, we analyze the incubation period of large flows, discuss LFD's tradeoffs, and present practical guidelines for configuring LFD. Finally, we compare LFD with closely related proposals [58, 57] to demonstrate that it outperforms prior work in terms of both efficiency and detection accuracy.

We consider a network link with a capacity of ρ, and a LFD detector with n counters. The counter threshold is β_{TH}. Once the value of a counter exceeds β_{TH}, the associated flow will be judged as a large flow and cut off immediately. Hence, the maximum value of each counter is $\beta_{TH} + \alpha$, where α is the maximum packet size. Table 5.1 summarizes the notations used in this section. We will discuss the relationship among parameters and how to set them in the Section 5.3.6.

Table 5.1: Table of Notations.

Network management parameters:		
ρ	$\triangleq$	Rate of link capacity
α	$\triangleq$	Maximum packet size
t_{upincb}	$\triangleq$	Upper bound of t_{incb} for any large flows
TH_ℓ	$\triangleq$	Low-bandwidth threshold
TH_h	$\triangleq$	High-bandwidth threshold
γ_ℓ, β_ℓ	$\triangleq$	Rate and burst for low-bandwidth threshold
γ_h	$\triangleq$	Rate for high-bandwidth threshold
Tunable parameters:		
n	$\triangleq$	Number of counters in LFD
β_{TH}	$\triangleq$	Threshold of counters($> \beta_\ell$)
Parameters that depend on tunable parameters:		
β_h	$\triangleq$	Burst for high-bandwidth threshold
β_Δ	$\triangleq$	$\beta_{TH} - \beta_\ell$
Other notations:		
$R(t_1, t_2)$	$\triangleq$	Average flow rate in $[t_1, t_2)$
t_{incb}	$\triangleq$	Incubation period of large flows
R_{NFN}	$\triangleq$	No-FN$_\ell$ rate
R_{NFP}	$\triangleq$	No-FP$_s$ rate

5.3.1 Large Flow False Negative Analysis

Theorem 9. ***No-FN$_\ell$ property.*** *LFD detects every flow violating the high-bandwidth threshold $TH_h(t) = \gamma_h t + \beta_h$ over a time window of length t, when $\gamma_h \geq R_{NFN} = \frac{\rho}{n+1}$ and $\beta_h \geq \alpha + 2\beta_{TH}$.*

Proof sketch: Firstly, we prove that LFD satisfies L3 in Theorem 8. According to Algorithm 1, the maximum value of each counter c_i is $\beta_{TH} + \alpha$, and there are at most n non-zero counters at any time. Also, given any valid synopsis $S = \{c_i\}$ we can construct a packet stream X consisting of c_i bytes for flow i and no space between packets, and by construction $S = \mathtt{Update}(S_0, X)$. Combining the above two arguments and the definition of the distance function, we conclude that $\mathsf{dis}(S_0, S) \leq \frac{\mathsf{size}(\mathcal{X})}{\rho} \leq \frac{(\beta_{TH}+\alpha)n}{\rho}$. That is, setting $\Delta = \frac{(\beta_{TH}+\alpha)n}{\rho}$ satisfies L3.

Next we prove that LFD satisfies L2 in Theorem 8 as well, when setting $\gamma'_h = R_{NFN} = \frac{\rho}{n+1}$ and $\beta'_h = \beta_{TH}$. We prove by contradiction and assume there were a flow f violating $\gamma'_h + \beta'_h$ in the landmark window model at time t but not being detected (i.e., $c_f < \beta_{TH}$). This assumption implies that more than $\gamma'_h t + \beta'_h - \beta_{TH}$ amount of flow f would have been canceled out[5] during the decrement step, or equivalently, more than $(\gamma'_h t + \beta'_h - \beta_{TH}) \cdot (n+1) = \gamma'_h t \cdot (n+1) = \rho t$ amount of traffic would have been canceled out. This statement, however, contradicts the setting where the maximum traffic for t units of time is ρt. Thus, f cannot escape from LFD, and L2 is satisfied by LFD.

Based on Theorem 8, we conclude that LFD satisfies A2 when $\gamma_h = \gamma'_h = \frac{\rho}{n+1}$ and $\beta_h \geq \beta'_h + \gamma_h \Delta = \beta_{TH} + \frac{\rho}{n+1} \frac{(\beta_{TH}+\alpha)n}{\rho} = \beta_{TH} + \frac{n}{n+1}(\beta_{TH} + \alpha)$. In particular, LFD catches every flow violating the threshold $TH_h(t) = \gamma_h t + \beta_h$ when $\gamma_h \geq R_{NFN}$ and $\beta_h \geq \alpha + 2\beta_{TH}$. That is, LFD catches all large flows in the arbitrary window model. □

5.3.2 Small Flow False Positive Analysis

As discussed in Section 5.2.1, no algorithm can satisfy A2 in Theorem 8 and L1 in Theorem 7 at the same time. Hence, rather than applying Theorem 7, we have to take a different approach in proving the no-FP$_s$ property.

To analyze LFD's no-FP$_s$ property, we consider how LFD increases and decreases its counter values. Firstly, let us examine all cases based on the types of incoming flows. We say a flow is old if it is stored in the counters currently; otherwise the flow is new.

1. When the incoming flows are virtual flows and there are l empty counters, in a time window t, the decrement is $\frac{\rho}{l+1} t$ on all counters, and the increment is 0. ($l = 0, 1, 2, 3, ..., n$)

[5]A packet byte is canceled out if it does not contribute to the corresponding counter.

2. When the incoming flows are new real flows and there is no empty counter, in a time window t, the decrement is ρt on all counters and the increment is 0 (which is the same as the first case when $l = 0$).

3. When the incoming flows are old real flows, or new real flows and there are some empty counters, in time interval t, the decrement is 0 and the increment is $\rho\, t$ on one counter.

Thus, in the first and second cases, when there are l empty counters in the detector, the decrement is always $\frac{\rho}{l+1} t$ in the interval of t. In the third case, the increment is always $\rho\, t$ on one counter in the interval of t. Finally, the increment and decrement cannot happen at the same time.

Lemma 10. *For any small flow f that complies with the low-bandwidth threshold (i.e., $TH_\ell(t) = \gamma_\ell\, t + \beta_\ell$), once the flow f is added to a counter at t_1, this counter will be always lower than β_{TH} after time $t_1 + t_{\beta_\ell}$ if the counter is occupied by the same flow as the flow f, where $t_{\beta_\ell} = \frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho}$.*

Detailed proofs are in Appendices 5.6.1 and 5.6.2.

Theorem 11. ***No-FP$_s$ property.*** *LFD will not catch any flow complying with the low-bandwidth threshold $TH_\ell(t) = \gamma_t + \beta_\ell$ for all time windows of length t, when $0 < \beta_\ell < \beta_{TH}$, $\gamma_\ell < R_{NFP}$, where $R_{NFP} = \frac{\beta_\Delta}{(n-1)\alpha+(n+1)\beta_\ell+(n+1)\beta_\Delta} \cdot \rho$.*

Proof sketch: According to Lemma 10, to avoid catching a small flow f, we can make the counter smaller than β_{TH} before t_{β_ℓ}. Hence, we choose a γ_ℓ to achieve $\gamma_\ell\, t_{\beta_\ell} + \beta_\ell < \beta_{TH}$. Then, $\frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho} < \frac{\beta_{TH}-\beta_\ell}{\gamma_\ell}$,

$$\Leftrightarrow \gamma_\ell < \frac{\beta_\Delta}{(n-1)\alpha + (n+1)\beta_\ell + (n+1)\beta_\Delta} \cdot \rho \tag{5.1}$$

Then, the theorem is proved; LFD guarantees that no small flow will be falsely caught. □

Interestingly, Theorem 11 shows that γ_ℓ approaches $\frac{\rho}{n+1}$ as β_Δ increases, but cannot go beyond $\frac{\rho}{n+1}$.

5.3.3 Relationship between Low-Bandwidth and High-Bandwidth Thresholds

Before the discussion, let us define two concepts:

Rate Gap: The ratio between γ_h and γ_ℓ (i.e. γ_h/γ_ℓ);

Burst Gap: The ratio between β_h and β_ℓ (i.e. β_h/β_ℓ).

Based on Theorems 9 and 11, the minimum rate gap is: $(\gamma_h/\gamma_\ell)_{min} = \frac{R_{NFN}}{R_{NFP}} = \frac{(n-1)\alpha+(n+1)(\beta_\ell+\beta_\Delta)}{\beta_\Delta(n+1)}$

Given $\beta_\Delta = \beta_{TH} - \beta_\ell$ and $n+1 \doteq n \doteq n-1$, we get

$$(\gamma_h/\gamma_\ell)_{min} \doteq 1 + \frac{2\alpha/\beta + 2}{\beta_h/\beta_\ell - (\alpha/\beta_\ell + 2)} \tag{5.2}$$

Thus, the minimum possible rate gap $(\gamma_h/\gamma_\ell)_{min}$ is mainly influenced by the burst gap β_h/β_ℓ. Equation (5.2) tells us (1) β_h/β_ℓ cannot be lower than $\alpha/\beta_\ell + 2$. (2) LFD only needs a low β_h/β_ℓ to achieve small enough $(\gamma_h/\gamma_\ell)_{min}$. For example, to achieve $(\gamma_h/\gamma_\ell)_{min} = 10$, we only need $\beta_h/\beta_\ell = 2.53$. (3) $(\gamma_h/\gamma_\ell)_{min}$ cannot be lower than 1. $(\gamma_h/\gamma_\ell)_{min}$ approaches to 1 as β_h/β_ℓ grows.

5.3.4 Incubation Period of Large Flows

To define the incubation period, we first consider a large flow that violates the high-bandwidth threshold over $[t_1, t_2)$, and the packet at t_a triggers the detection. Because of LFD's no-FN$_\ell$ property, $t_a \le t_2$. The incubation period is defined as $t_a - t_1$, which represents the time duration for which the large flow remains under the radar. We bound the incubation period as follows.

Theorem 12. *For the flow f which violates $TH_h(t)$ over some time window $[t_1, t_2)$, if its average rate $R(t_1, t_a)$ is larger than R_{atk} in time interval of $[t_1, t_a)$ (R_{atk} is a constant rate larger than $R_{NFN} = \frac{\rho}{n+1}$), then f's incubation period is bounded by*

$$t_{incb} < \frac{\alpha + 2\beta_{TH}}{R_{atk} - \frac{\rho}{n+1}}. \tag{5.3}$$

Proof sketch: Because $R(t_1, t_a) > R_{atk}$, intuitively the t_{incb} of flow with an average rate of $R(t_1, t_a)$ must be shorter than the t'_{incb} of flow with rate of R_{atk}. That is, $t_{incb} < t'_{incb}$.

Assume a flow f' with rate R_{atk} will violate $TH_h(t)$ over time window $[t'_1, t'_2)$, then

$$\begin{aligned} R_{atk}(t'_2 - t'_1) &= \frac{\rho}{n+1}(t'_2 - t'_1) + \alpha + 2\beta_{TH} \\ \Rightarrow t_{incb} < t'_{incb} = t'_a - t'_1 \le t'_2 - t'_1 &= \frac{\alpha + 2\beta_{TH}}{R_{atk} - \frac{\rho}{n+1}} \end{aligned} \tag{5.4}$$

Thus, the theorem is proved. □

From Theorem 12, the bound of the incubation period decreases as R_{atk} increases. In other words, if R_{atk} is fixed, the bound of the incubation period decreases with increasing n, which implies we can reduce the upper bound by adding extra counters. To guarantee detection of flows whose rate is over $R_{atk}(R_{atk} > \frac{\rho}{n+1})$, the minimum number of counters is $\frac{\rho}{R_{atk}} - 1$, and the upper bound on the incubation period can be lowered significantly by adding a few counters. The details will be discussed in Section 5.3.6.

5.3.5 Tradeoff Analysis

We discuss three tradeoffs in LFD: (1) memory consumption (i.e., the number of counters) vs. the rate gap, (2) the rate gap and burst gap, and (3) the rate gap and the upper bound on the incubation time.

First, since the rate gap can be expressed as $\gamma_h/\gamma_\ell > R_{NFN}/\gamma_\ell = \frac{\rho/\gamma_\ell}{n+1}$, we can see the rate gap decreases with increasing n. Second, Equation (5.2) shows that the minimum rate gap γ_h/γ_ℓ is mainly influenced by β_h/β_ℓ, namely the burst gap, and the minimum rate gap decreases as the burst gap increases. Finally, Theorem 12 shows that a large burst gap results in a long incubation period. Hence, a small rate gap results in a big burst gap and a high incubation period.

5.3.6 How To Engineer The Detector

To engineer our detector, we first need to be clear on what parameters are known before starting to design it. Usually, users want a detector for a specific link capacity ρ, to protect small flows which comply with the low-bandwidth threshold: $TH_\ell(t) = \gamma_\ell t + \beta_\ell$, and to detect attack flows that violate the high-bandwidth threshold: $TH_h(t) = \gamma_h t + \beta_h$. However, as discussed in Section 5.3.5, there is a tradeoff between the rate gap and burst gap, so their requirements cannot be both fulfilled. Thus, we choose to satisfy the rate requirement of γ_h first, and then set β_h according to γ_h, as it is more important to limit the flow rate than the burst size. Furthermore, since we want to minimize the incubation period of large flows, there is a requirement on the upper bound of the incubation period, t_{upincb}.

We set $\beta_h = \alpha + 2\beta_{TH}$ and $\gamma_h > \frac{\rho}{n+1}$ to guarantee no FN$_\ell$ according to Theorem 9. Since $\beta_{TH} = \beta_\ell + \beta_\Delta$, we only need to decide the number of counters n and β_Δ. Hence, the problem can be simplified as follows. Given ρ, γ_ℓ, β_ℓ, γ_h, α, and t_{upincb}, we aim to calculate n and β_Δ such that the parameters satisfy the constraints in Theorems 9, 11 and 12.

5.4 Evaluation

In this section, we evaluate LFD using both theoretical and experimental evaluations and compare the results with two closely related proposals, which we refer to as FMF [58] and AMF [57], to demonstrate that LFD performs better than prior work in terms of both exactness outside an ambiguity region and efficiency. The results of experiments using real and synthetic traffic traces are consistent with the analysis in our theoretical evaluation.

5.4.1 Theoretical Comparison

Multistage filters. *Fixed-window-based Multistage Filters* (FMF) identify large flows in a fixed measurement interval. A FMF consists of parallel stages, each of which is an array of counters initialized to zeros at the beginning of a measurement interval. Each stage is assigned a hash function that maps a packet's flow identifier to a counter in the stage. For each incoming packet, its flow identifier is hashed to one counter in each hash stage, and the counter value increments by the size of the packet. A flow is considered a large flow if all of its corresponding counters exceed a pre-specified threshold.

Arbitrary-window-based Multistage Filters (AMF) identify large flows over arbitrary windows. To work in the arbitrary window model, AMF replaces each counter in FMF with a leaky bucket of a bucket size u and a drain rate r. A flow is considered a large flow if the corresponding leaky buckets are all violated.

Performance Comparison. Table 5.2 presents a numerical example of LFD, FMF and AMF, where the high-bandwidth rate is 1% of the link capacity, and the low-bandwidth rate is 0.1% of link capacity. The results of FMF and AMF are derived based on the authors' original analysis that assumes a specific number of active flows for the input traffic.

Table 5.2: A numerical example given the requirements in Section 5.3.6. γ_h is 1% of the link capacity, and γ_ℓ is 0.1% of link capacity

Scheme	# of counters	FP_s rate	FN_ℓ rate
LFD	101	0	0
FMF	101/1000	*no guarantee* / $\leq$ 0.04	0*
AMF	101/2000	*no guarantee* / $\leq$ 0.04	0

*FMF's FN_ℓ rate is above 0 in the arbitrary window model.

Table 5.3: Comparison of three schemes.

Scheme	FP_s	FN_ℓ	Memory	Input Traffic
LFD	no	no	low	independent
FMF	yes	yes	high	dependent
AMF	yes	no	high	dependent

LFD outperforms the other two approaches in several aspects: 1) LFD guarantees no false detection of small flows, whereas they cannot. Even using tens of times of extra storage space, the FMF and AMF still have error rates as high as 0.04. 2) LFD and AMF can detect all large flows. However, FMF has FN_ℓ on bursty flows. 3) LFD requires much less memory compared with multistage filters. 4) LFD's performance is independent of input traffic, because the error rate is always zero, while multistage filters require more stages as the number of active flows increases so as to keep the same false positive. Table 5.3 summarizes the comparison, which suggest LFD is exact outside the ambiguity region and efficient comparing to prior works. And we have detailed comparison in the Section 5.4.2 and 5.4.3 , in which concrete examples are discussed to support our theoretical results above.

Although LFD presents several advantages compared with multistage filters, LFD cannot estimate the size of a detected flow, which multistage filters achieve.

5.4.2 Experiment Settings

Datasets. Table 5.4 summarizes the characteristics of the two datasets used in the experiments. The Federico II dataset contains traces collected at the TCP port 80 of a 200 Mbps link [40, 41, 9]. The CAIDA dataset contains anonymized passive traffic traces from CAIDA's equinix-sanjose monitors on 10 Gbps backbone links [7]. For each dataset, we use the first 30 seconds for experiments. We define flows based on the source and destination IP addresses.

Table 5.4: Dataset Information.

Dataset	Link capacity	Avg link rate	# of flows	Avg flow size
Federico II	$200Mbps$	$1.85MB/s$	2911	$19.9KB$
CAIDA	$10Gbps$	$279.65MB/s$	2517099	$3.3KB$

Table 5.5: Parameters of Experiment Environment.

Dataset	γ_h	β_h	γ_ℓ	β_ℓ	ρ	α	link status	β_{TH}	n	t_{upincb}
Federico II	$250KB/s$	$15.5KB$	$25KB/s$	$6072B$	$25MB/s$	$1518B$	congested/non-congested	$6991B$	107	$0.8370sec$
CAIDA	$12.5MB/s$	$15.4KB$	$1.25MB/s$	$6072B$	$1.25GB/s$	$1518B$	non-congested	$6925B$	100	$0.1242sec$

Attack scenarios. In the experiments, we are interested to know LFD's performance compared with FMF and AMF in the face of some common attacks.

In particular, we generate attack flows using two simple strategies—*flooding attacks* and *Shrew DoS attacks* [93, 71]—and then mix real traces with artificially generated attack flows to simulate an attack environment. In a flooding attack, the adversary sends high-rate flows with a specified rate γ_{large} (e.g. γ_h). Each high-rate flow is generated as follows. We randomly choose a 1-second time slot within the 30-second stream as the first second of the flow. Starting from that second, we randomly generate $\gamma_{large}/packetSize$ packets in each 1-second interval to make the flow size in each interval equal to that specified rate. In this experiment, we set the packet size to 1518 bytes, the maximum packet size. In a Shrew attack, the attacker sends periodic bursts in an attempt to cut off TCP traffic by exploiting TCP's congestion control mechanism. To generate a bursty flow with a period T, burst duration L, and bursty flow rate γ_{burst}, we randomly choose a time point from $[0, 29)$ second as the start time, and then randomly generate $\gamma_{burst} \cdot L$ packets in each L-length burst that occurs every T seconds. We then evaluate (1) how many malicious large flows can evade detection, and (2) how many small benign flows are falsely caught because of these coexisting attack flows.

We configure LFD based on the guidelines in Section 5.3.6, such that it can detect large flows violating $TH_h(t) = \gamma_h t + \beta_h$, where $\gamma_h = 1\%\rho$ and $\beta_h = 2\beta_{TH} + \alpha$ are determined based on Equation 5.21. We also consider small flows that comply with $TH_\ell(t) = \gamma_\ell t + \beta_\ell$, where $\beta_\ell = 6072$ bytes and $\gamma_\ell = 0.1\%\rho$. Also, we require t_{upincb} to be smaller than 1 sec. Table 5.5 summarizes the value of each parameter used in our experiments. In a "non-congested link" setting, a fixed number of attack flows are mixed with the real trace. We also consider a "congested link" setting, where we fill the link with attack flows for the small dataset (i.e., the Federico II dataset) only. We leave it as future work to scale our attack flow generation tool to work for larger datasets.

To configure the two multistage filters (FMF and AMF), we set FMF's window size to $1sec$, number of stages $d = 2$, number of counters in each stage $b = 250$, threshold of FMF $T = \gamma_h \cdot 1sec$, threshold of AMF $u = \beta_h$, and drain rate $r = \gamma_h$. We are also interested in the performance of FMF and AMF when their memory is as small as LFD's. Hence, we run additional experiments in which the number of counters in each stage is 55. The details of these values are shown in the Table 5.6.

Then, for each experiment environment, we design two sets of experiments to test the performance of these three filters in the presence of flooding attacks and Shrew attacks. We repeat each experiment for 10 times and present the average. In the case of flooding attacks, we randomly generate k_1 attack flows for each attack rate (the k_1 and k_2 will be explained later). In the case of Shrew attacks, we randomly generate k_2 bursty flows with $1.2 * \gamma_h$ burst rate and $1sec$ period for each burst duration L. We set $k_1 = k_2 = 50$ for the non-congested link setting, and set the k_1 and k_2 as large as possible to congest the link in the congested-link setting.

Table 5.6: Multistage Filter Parameters

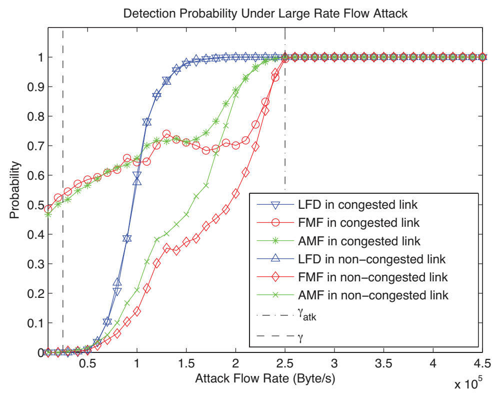
Dataset	$b * d$	T	u	r
Federico II	$55 * 2, 250 * 2$	$250KB$	$15.5KB$	$250KB/s$
CAIDA	$55 * 2, 250 * 2$	$12.5MB$	$15.4KB$	$12.5MB/s$

Evaluation metrics. We consider three evaluation metrics: detection probability, false positive probability of small flows, and incubation period. Detection probability is the probability to successfully detect a generated flow. False positive probability of small flows is the probability to wrongly catch a small flow when the link is attacked by attack flows of a certain rate. Incubation period represents the time needed to catch a generated attack flow since it is generated.

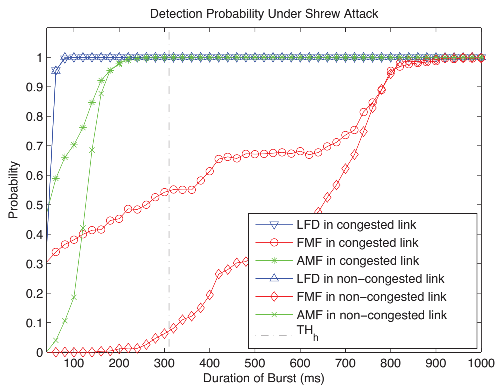
5.4.3 Experimental Comparison

Since the results for the two datasets are similar, we omit CAIDA's results due to the space constraints.

Figure 5.4a and 5.4b shows the detection probability in the face of different types of attack flows. We focus on the scenario of using $55 * 2$ counters in FMF and AMF, as the results of $250 * 2$ counters are similar.

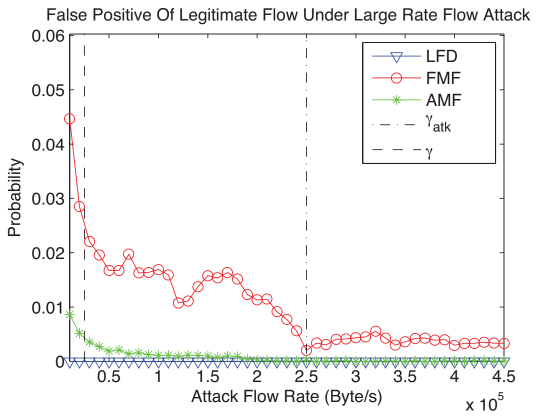


(a) Flooding DoS Attack

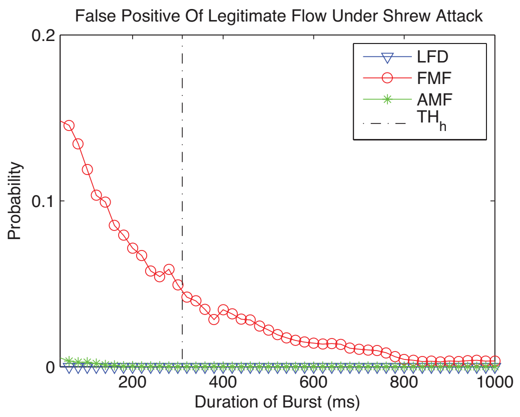


(b) Shrew DoS Attack

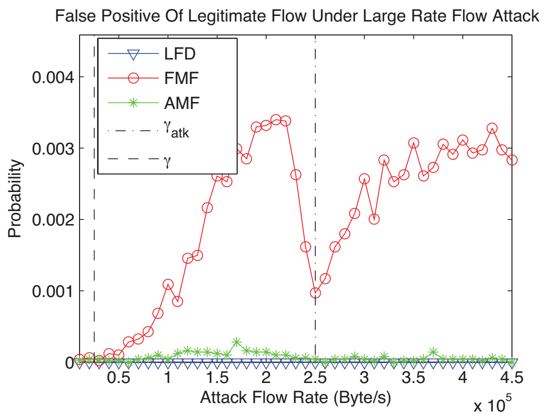
Figure 5.4: Detection Probability in Experiment with $55 \cdot 2$ counters in Multistage Filter.



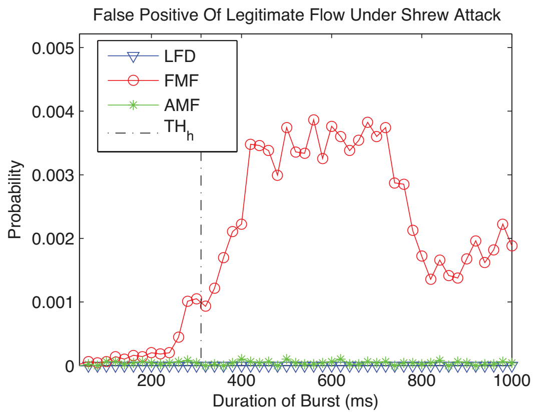
(a) 55*2 counters - Congested Link



(b) 55*2 counters - Congested Link



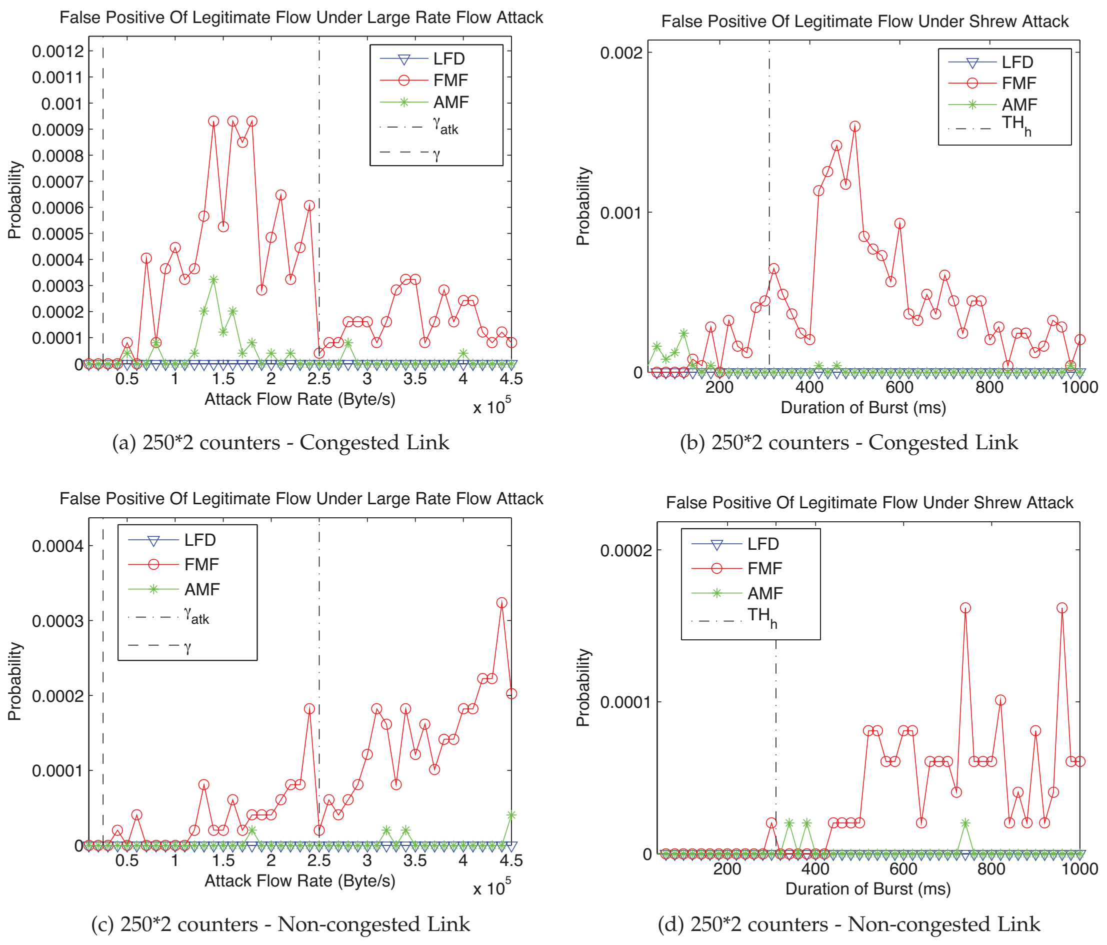
(c) 55*2 counters - Non-congested Link



(d) 55*2 counters - Non-congested Link

Figure 5.5: False Positive of Small Flows with $55 \cdot 2$ counters.

In Figure 5.4b, the TH_h line indicates whether a bursty flow exceeds the high-bandwidth threshold. The results show that LFD detects attack flows that are large with a 100% detection probability, which confirms



(a) 250*2 counters - Congested Link

(b) 250*2 counters - Congested Link

(c) 250*2 counters - Non-congested Link

(d) 250*2 counters - Non-congested Link

Figure 5.6: False Positive of Small Flows with 250 · 2 counters.

Theorem 9. Figure 5.4b shows that FMF cannot catch most of the Shrew flows. Moreover, LFD can catch most of the attack flows in the medium-flow area (between $TH_\ell(t)$ and $TH_h(t)$).

Figure 5.5a to Figure 5.6d show the results of FP_s rates. While LFD has zero FP_s in any case as expected, Figure 5.5a to Figure 5.5d shows that both FMF and AMF have high FP_s rates in both attack scenarios when using very limited memory as LFD. That is, the attacker can successfully incriminate benign small flows. Worst yet, when the link is congested by attack flows, the FP_s rate can be as high as 4% for FMF and 1% for AMF under flooding attacks, and for FMF, the FP_s rate is also extremely high under Shrew attacks. An interesting observation is that, in Figure 5.5a, both FMF and AMF have a higher FP_s rate when the link is congested by malicious small flows. As Figure 5.6a to 5.6d show, using more counters in FMF and AMF can reduce, but not eliminate, the FP_s rates. The results of the CAIDA dataset exhibit similar trends.

Also, in our experiments, LFD always produces similar results no matter whether the link is congested.

In contrast, the results of AMF and FMF are much different between the congested-link version and non-congested-link version. This supports the conclusion in Section 5.4.1 that AMF and FMF rely on the number of active flows but LFD does not. This advantage makes LFD stable in any networking environment.

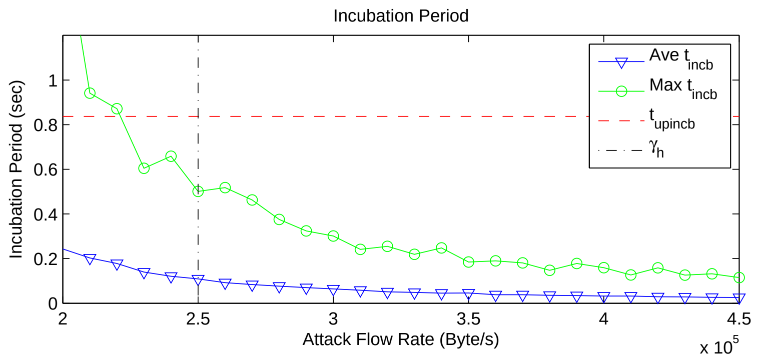


Figure 5.7: Incubation Period.

Figure 5.7 describes the maximum and average incubation period of high-rate flows with different rates in flooding attacks. We can find that the maximum incubation period for flows whose rate is over γ_h is always below the theoretical upper bound for the incubation period, t_{upincb}, which supports Theorem 12. Moreover, the average incubation period is much lower than the theoretical upper bound, which shows LFD's incubation period is much shorter in practice.

5.5 Summary

LFD is a deterministic streaming algorithm that robustly catches all large flows and protects all small flows regardless of the traffic distribution. The core ideas differentiating LFD from prior work are that it (1) monitors flows over arbitrary windows and (2) provides exactness outside an ambiguity region. One future direction is to explore the design space of large-flow algorithms in the arbitrary window model by applying the no-FP and no-FN theorems to existing landmark-window-based algorithms. Another interesting future work is to formally examine the robustness of LFD and prior algorithms against malicious inputs. We believe that LFD can aid emerging applications such as detecting flooding attacks by bursty flows [93] and enforcing QoS-based SLA compliance [158], which require robust monitoring for high assurance.

5.6 Appendix

5.6.1 Lemma 13 and Proof Sketch

Lemma 13. *In any time interval $[t_1, t_2]$ with k large flows occupying k counters from the beginning time t_1 to the ending time t_2, if all the other counters (i.e., counters except the ones occupied by large flows) are empty at beginning time t_1 and ending time t_2, then the decrement of all the counters is $\frac{(t_2-t_1)-t_{lrg}}{n+1-k}\rho$, where t_{lrg} is the sum of time for which these k large flows are sending packets.*

Proof sketch: In $[t_1, t_2]$, when these large flows occupy the link for t_{lrg} time, then the link is occupied by certain real flows F or virtual flows for $t_2 - t_1 - t_{lrg}$ time. There is no assumption about the flows in F, except they should ensure that there are $n-k$ empty counters at beginning time t_1 and ending time t_2. During the time of $t_2 - t_1 - t_{lrg}$, the counters are either increased by flows in F or decreased by flows in F or virtual flows.

We divide the total decrement dec into many small decrements dec_i. dec_i happens in time interval $t_{i,dec}$, and the number of counters occupied by flows in F is x_i during $t_{i,dec}$.

Because the other $n-k$ counters are empty at the beginning and the ending, when there is a decrement dec_i for each counter, then there must be x_i increment inc_i on x_i non-empty counters. Therefore, all the decrements dec_i in these $n-k$ counters have a counterpart of x_i increment inc_i lasting for $t_{i,inc}$ time. dec_i and inc_is may be apart in the time domain, but for a decrement dec_i there must be x_i number of inc_is, such that $inc_i = dec_i$.

According to the three ways of counter decreasing and increasing, given the number of empty counters is $l = n - k - x_i$, the dec_i and inc_i in $t_{i,dec}$ are:

$$dec_i = \frac{\rho}{n+1-k-x_i} \cdot t_{i,dec} \tag{5.5}$$

$$inc_i = \rho \cdot t_{i,inc} \tag{5.6}$$

Then, according to $inc_i = dec_i$ and (5.5,5.6)

$$\Rightarrow \begin{cases} t_{i,dec} = \dfrac{(n+1-k-x_i)\cdot dec_i}{\rho} \\ t_{i,inc} = \dfrac{inc_i}{\rho} = \dfrac{dec_i}{\rho} \end{cases} \tag{5.7}$$

At any time point in $t_2 - t_1 - t_{lrg}$, counters are either increasing or decreasing, which means

$$t_2 - t_1 - t_{lrg} = \sum_i (x_i \cdot t_{i,inc} + t_{i,dec}) \tag{5.8}$$

Then, according to (5.7,5.8), we can get

$$t_2 - t_1 - t_{lrg} = \sum_i (x_i \cdot \frac{dec_i}{\rho} + \frac{(n+1-k-x_i) \cdot dec_i}{\rho}) \tag{5.9}$$

$$= \frac{dec(n+1-k)}{\rho} \Rightarrow dec = \frac{(t_2 - t_1) - t_{lrg}}{n+1-k}\rho \tag{5.10}$$

Therefore, during $[t_1, t_2]$ the decrement of all the counters is $\frac{(t_2-t_1)-t_{lrg}}{n+1-k}\rho$, and this lemma is proved. □

5.6.2 Proof Sketch of Lemma 10

Proof sketch: WLOG, we assume flow f is associated with a counter at $t_1 = 0$, and in $[0, t_{ocp}]$, flow f always occupies this counter. Then, intuitively, in $[0, t_{ocp}]$, the cases to have minimum decrement dec_{min} on this counter are: 1) at time 0 all the counters are empty, and 2) at time t_{ocp}, except the counter of flow f, all other counters have the maximum value $\alpha + \beta_{TH}$. Because the remaining values in the counter will cost extra time t_{inc} for increasing these counters, then according to the Lemma 13, the $t_2 - t_1$ in Lemma 13 is smaller and the decrement is smaller. Therefore, in the case mentioned above, the decrement is minimized. According to Lemma 13, in this case $t_2 - t_1 = t_{ocp} - t_{inc}$, $k = 1$, then the minimum decrement is:

$$dec_{min} = \frac{t_{ocp} - t_{inc} - t_{lrg}}{n}\rho \tag{5.11}$$

where $t_{inc} = \frac{(n-1)(\beta_{TH}+\alpha)}{\rho}$.

Since f complies with $TH_\ell(t)$, $t_{lrg} < \gamma_\ell/\rho \cdot t_{ocp} + \frac{\beta_\ell}{\rho}$.

$$\Rightarrow dec_{min} > \frac{t_{ocp}(1-\gamma_\ell/\rho)}{n}\rho - \frac{(\beta_{TH}+\alpha)(n-1)+\beta_\ell}{n} \tag{5.12}$$

$$\Leftrightarrow dec_{min} > \gamma_\ell\, t_{ocp} + \frac{t_{ocp}(1-(n+1)\frac{\gamma_\ell}{\rho})}{n}\rho - \frac{(\beta_{TH}+\alpha)(n-1)+\beta_\ell}{n} \tag{5.13}$$

When $t_{ocp} > t_{\beta_\ell} = \frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho}$,

$$\Rightarrow dec_{min} > \gamma_\ell\, t_{ocp} + \frac{(n-1)\alpha+(n+1)\beta_\ell}{n}\rho - \frac{(\beta_{TH}+\alpha)(n-1)+\beta_\ell}{n} \tag{5.14}$$

$$\Rightarrow \gamma_\ell\, t_{ocp} + \beta_\ell - dec_{min} < \beta_{TH} \tag{5.15}$$

Because flow f complies with $TH_\ell(t)$, its counter value is smaller than $t_{ocp} + \beta_\ell - dec_{min}$. Therefore, the counter is smaller than β_{TH} after t_{β_ℓ}. □

5.6.3 Engineering The Parameters

We give a detailed solution and analysis to the problem defined in Section 5.3.6. The problem can be expressed by the inequality set (5.16):

$$\begin{cases} \dfrac{\alpha + 2\beta_{TH}}{\gamma_h - \frac{\rho}{n+1}} < t_{upincb} \\ \dfrac{\beta_\Delta}{\alpha(n-1) + (n+1)\beta_\ell + (n+1)\beta_\Delta} \cdot \rho > \gamma_\ell \\ \dfrac{\rho}{n+1} < \gamma_h \end{cases} \tag{5.16}$$

$$\Leftarrow \begin{cases} \dfrac{2(\alpha + \beta_{TH})}{\gamma_h - \frac{\rho}{n+1}} \le t_{upincb} \\ \dfrac{\beta_\Delta}{\alpha + \beta_\ell + \beta_\Delta} \cdot \dfrac{\rho}{n+1} \ge \gamma_\ell \\ \dfrac{\rho}{n+1} < \gamma_h \end{cases} \tag{5.17}$$

$$\Leftrightarrow \begin{cases} \beta_\Delta \le \dfrac{t_{upincb}(\gamma_h - \frac{\rho}{n+1}) - 2(\alpha + \beta_\ell)}{2} \\ \beta_\Delta \ge \dfrac{\gamma_\ell(\alpha + \beta_\ell)}{\frac{\rho}{n+1} - \gamma_\ell} \\ n > \dfrac{\rho}{\gamma_h} - 1 \end{cases} \tag{5.18}$$

$$\Rightarrow \begin{cases} \dfrac{\gamma_\ell(\alpha + \beta_\ell)}{\frac{\rho}{n+1} - \gamma_\ell} \le \dfrac{t_{upincb}(\gamma_h - \frac{\rho}{n+1}) - 2(\alpha + \beta_\ell)}{2} \\ n > \dfrac{\rho}{\gamma_h} - 1 \end{cases} \tag{5.19}$$

Then we can assert that there must exist a solution pair of (n, β_Δ) which fulfills inequality set (5.18), if and only if there is a n satisfying $n_{min} \le n \le n_{max}$, where

$$\begin{cases} n_{min} = \lceil \rho / \dfrac{M + \sqrt{M^2 - 4\gamma_h\gamma_\ell}}{2} \rceil - 1 \\ n_{max} = \lfloor \rho / \dfrac{M - \sqrt{M^2 - 4\gamma_h\gamma_\ell}}{2} \rfloor - 1 \\ M = \gamma_h + \gamma_\ell - \dfrac{2(\alpha + \beta_\ell)}{t_{upincb}} \ge 0 \end{cases} \tag{5.20}$$

Then, we obtain the solution space, as Figure 5.8 illustrates. In this figure, the (n, β_Δ) solution pairs are in the space between the two lines of the lower bound curve and the upper bound curve. Note that the inequality sets (5.16) and (5.18) are not totally equal, so there may be additional solutions outside this space.

According to the inequality set (5.18), the lower bound of β_Δ is $\beta_{\Delta min} = \frac{\gamma_\ell(\alpha+\beta_\ell)}{\frac{\rho}{n+1} - \gamma_\ell}$. And we can see $\beta_{\Delta min}$ increases with n, as Figure 5.8 shows. Since we can reduce memory consumption and the burst gap by using a smaller n and β_Δ, we choose

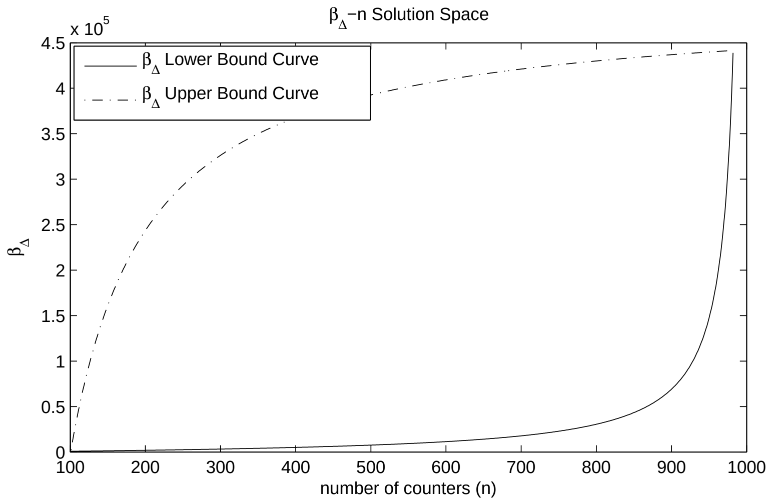


Figure 5.8: Curve of the lower bound of β_ℓ. ($\gamma_\ell = 100,000$ byte/s $\gamma_h = 1,000,000$ byte/s, $\rho = 100,000,000$ byte/s, $\alpha = 1518$ bytes, $\beta_\ell = 6072$ bytes, $t_{upincb} = 1$ sec.)

$$\begin{cases} n = n_{min} = \lceil \rho / \frac{M + \sqrt{M^2 - 4\gamma_h \gamma_\ell}}{2} \rceil - 1 \\ \beta_{TH} = \beta_\ell + \beta_{\Delta\, min} = \beta_\ell + \frac{\gamma_\ell(\alpha + \beta_\ell)}{\frac{\rho}{n_{min}+1} - \gamma_\ell} \end{cases} \tag{5.21}$$

as our final answer to this design problem.

We give a numerical example showing how to configure LFD based on the result above. Suppose the administrator of the detector chooses $\gamma_\ell = 100KB/s$, $\gamma_h = 1MB/s$, $\rho = 100MB/s$, $\alpha = 1518$ bytes, $\beta_\ell = 6072$ bytes, and $t_{upincb} = 1$ sec. Then using equation (5.21), we choose $n = 101$ and $\beta_\Delta = 863$ Byte. With these two parameters, the incubation period is 0.7848 sec which is smaller than $t_{upincb} = 1$ sec, and the no false positive rate is 100450 Byte/s which is larger than $\gamma_\ell = 100000$. The ratio between no false negative rate $\frac{\rho}{n+1}$ and low-bandwidth threshold rate γ_ℓ is $\frac{\rho}{n+1}/\gamma_\ell = 9.80$. The results show that to react quickly to large flows and obtain a small rate gap, the detector only needs a small number of extra counters comparing with the minimum number of required counters (i.e. $\frac{\rho}{\gamma_h} - 1 = 99$) and a low burst gap.

We obtain this particular solution by choosing the minimum n and minimum β_Δ. We can also solve the inequality set (5.16) for different requirements, such as minimizing the rate gap between $\frac{\rho}{n+1}$ and $\frac{\beta_\Delta}{\alpha(n-1)+(n+1)\beta+(n+1)\beta_\Delta} \cdot \rho$.

However, there may be no solution to the inequality set (5.18) for any given $\rho, \gamma_\ell, \beta_\ell, \gamma_h, \alpha, t_{upincb}$. To make it solvable, we need to make sure $M^2 - 8\gamma_h\gamma_\ell$ and M are not negative in the inequality set (5.20). Namely,

$$\gamma_h + \gamma_\ell - \frac{2(\alpha + \beta_\ell)}{t_{upincb}} \geq \sqrt{4\gamma_h\gamma_\ell} \tag{5.22}$$

$$\Leftarrow t_{upincb} \geq \frac{2(\alpha + \beta_\ell)}{\gamma_h + \gamma_\ell - 2\sqrt{\gamma_h\gamma_\ell}} \tag{5.23}$$

Moreover, according to Section 5.3.3, $\gamma_h > \gamma_\ell$ is necessary to make the inequality set (5.18) solvable.

Chapter 6

Lightweight Anonymity and Privacy

To improve Internet availability, previous chapters explore how to provide waiting time and bandwidth guarantees despite DDoS attacks. In this chapter, we turn our focus to study efficient defense against selective dropping, which is another major threat hindering the availability of the Internet.

In selective dropping, the attacker controls some devices, such as routers or firewalls, on the communication path, and selectively delays or blocks a subset of traffic. To mitigate selective dropping, our core observation is that *making flows indistinguishable from each other prevents the attacker from reacting selectively in the first place*. Technically, traffic indistinguishability forces a smart, selective-dropping attacker to become a dumb attacker that can at best block traffic at random.

Encryption and anonymization are common techniques to make traffic indistinguishable based on content and network identifiers, respectively [16, 17, 144, 56]. Concealing network identifiers is more challenging than concealing other information in the packets, as routers require network identifiers to forward packets. As a result, while many lightweight end-to-end encryption schemes exist (e.g., IPsec), anonymization schemes tend to be inefficient in terms of latency and computational overhead.

In this chapter, we present a lightweight protocol to hide the network identifiers from a remote attacker on the communication path. Consequently, this protocol prevents the attacker from discriminating traffic based on the topological information such as source and destination addresses.

Motivation: Existing anonymity systems introduce intolerable latency. Staying anonymous in today's Internet requires anonymous overlay systems, such as Tor [47], to conceal the communicating endpoint's IP address, as it can reveal the end-user's identity and location [119]. Such overlay systems attempt to facilitate anonymous communication using layer-encrypted packets traveling through indirect routes. However, this results in additional latency due to long end-to-end path length and cryptographic opera-

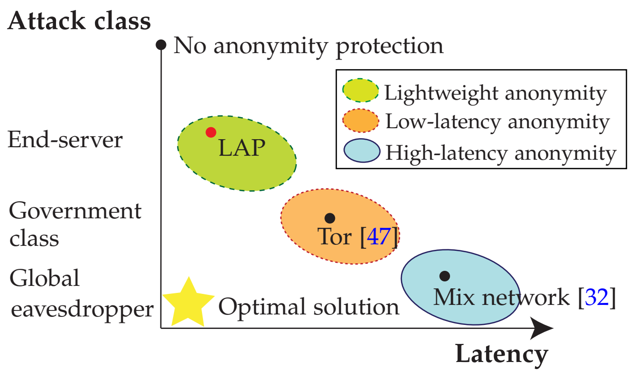


Figure 6.1: The design space of anonymous schemes.

tions indirectly traveling through three Tor relays would be approximately four times slower than traveling along a non-dilated path. Moreover, Tor relays are constantly overloaded [48], further worsening the latency and throughput. Measurements show that the average time to fetch an HTTP header using Tor is 4.04s—ten times higher than fetching it without Tor [128]. Although privacy-anxious users may tolerate seconds of latency for strong privacy, users desiring an intermediate level of privacy for default protection of daily online activities (e.g., prevent websites from tracking them for behavioral advertising[1]) may be impatient to wait.[2]

Despite existing work that attempts to protect end-users' anonymity [47, 136, 32], it still remains a challenge to provide an intermediate level of anonymity and privacy protection without introducing much latency. In this chapter, our main goal is to bridge the chasm between systems that provide strong anonymity with high latency and systems that support no anonymity with zero latency, and explore how to support *lightweight anonymity and privacy* that is efficient enough to protect all traffic. Note that those end-users who want an intermediate level of privacy primarily desire to remain anonymous from servers such that servers cannot track their behavior. This implies that guaranteeing the end-user's anonymity and privacy against a single remote entity rather than a strong, global attacker may be a suitable relaxation of the attacker model to gain higher efficiency.

Our solution: Lightweight anonymity and privacy with near-optimal latency under a relaxed attacker model. We propose a new setting that we call Lightweight Anonymity and Privacy (LAP-setting for short) for private and anonymous communication in the Internet with the following properties:

[1]Users might enable the DO-NOT-TRACK option supported by most mainstream browsers. However, a recent study [105] has shown that this mechanism is hard to use due to configuration complexity and provides no guarantee as it depends on the self-regulation of online organizations.

[2]Studies have shown that online users are sensitive to waiting time: Amazon's sales dropped by 1% for every 100ms increase in page load time, and Google's ad revenue decreased by 20% for a 500ms increase in search result display time [88].

- **Low-stretch anonymity:** Packets for anonymous and private communication should travel through near-optimal routes such that the increase in the number of Autonomous Domains (ADs) normalized over the original path length is low.

- **Relaxed attacker model:** An intermediate level of privacy can be achieved with sender and receiver anonymity and location privacy. Hence, we relax the strong attacker model (e.g., global or government-class attackers) considered by existing anonymity systems.

As Figure 6.1 shows, our aim is to address a relaxed attacker model (e.g., end-server attack) with near-optimal latency while existing work addresses stronger attacker models (e.g., government class or global eavesdropper) with higher latency. Although low-latency designs are shown to be inherently vulnerable to a global eavesdropper, some users who trust their local ISPs can achieve much higher efficiency under the LAP-setting.

Our mechanism, *Lightweight Anonymity and Privacy (LAP)*, is an efficient and practical network-based solution featuring lightweight path establishment and efficient communication. LAP attempts to enhance anonymity by obscuring an endhost's topological location, based on two building blocks: packet-carried forwarding state and forwarding-state encryption.

- **Packet-carried forwarding state:** Each packet carries its own forwarding state such that ADs can determine the next hop from the packet without keeping local per-flow state.

- **Forwarding-state encryption:** Existing anonymity systems require entire packets to be decrypted/encrypted as they travel using shared keys between the sender and intermediate relays. In contrast, LAP allows each AD to use a secret key (known to the AD only) to encrypt/decrypt forwarding information in packet headers. As a result, an AD's forwarding information can be hidden from all other entities while a LAP packet remains the same at each hop.

LAP is extremely lightweight in the sense that (i) it introduces minimal overhead over non-anonymous packets in terms of latency and computational overhead on routers, (ii) it does not require any per-flow state to be stored on routers, and (iii) no separate keys are required to be set up with routers. In addition to its performance advantages, LAP's unique design provides two additional merits. First, LAP supports different privacy levels such that an endhost can trade privacy for improved performance. Second, LAP is a generic design that can work with a wide range of routing protocols, which includes the inter-domain routing protocol BGP and new proposals such as SCION [178] and MobilityFirst [2]. Furthermore, we show that LAP fits especially well with proposed routing protocols that support packet-carried forwarding state, such as SCION and ICING [126].

In this chapter, we focus on network-based solutions, where users and locations can be identified through IP addresses. While most current endhost tracking is implemented via cookies [55] and applications may as well leak identifiable information such as email addresses or browser configurations, IP addresses have been used as an alternate identifier when such auxiliary information like cookies is unavailable.[3] Hence, a complete solution for anonymous communication must integrate network-layer techniques with mechanisms for other layers, as recognized by previous network-based proposals [136, 108].

Contributions.

1. We explore the design space of anonymous protocols in the context of a relaxed adversary model.

2. We propose Lightweight Anonymity and Privacy (LAP), an efficient network-based solution that enables lightweight path establishment and efficient forwarding.

3. We evaluate LAP's security and performance advantages. Our systematic analysis and the evaluation of our software implementation confirm that LAP can improve anonymity with low performance overhead.

6.1 Problem Definition

We study how to camouflage an endhost's topological location (i.e., potential origin within a given topological neighborhood) in a network architecture to enhance anonymity and location privacy in a practical manner. More specifically, we study how to design an anonymous forwarding protocol that can protect the identities and locations of endhosts from a weaker yet practical adversary, while demanding minimal increase in latency. We do not claim to achieve complete anonymity, but rather focus on providing an intermediate level of anonymity.

In this section, we scope our problem in terms of desired properties, assumptions, and threat model.

6.1.1 Desired Privacy Properties

Sender/receiver anonymity. Anonymity can be viewed as being unidentifiable within a set of subjects (e.g., users), also known as an *anonymity set* [131]. This implies that a sender or a receiver can achieve stronger anonymity if its identity is hidden in a larger anonymity set [19].[4] As a result, an attacker

[3]British Telecom Phorm PageSense External Validation report.
`http://www.wikileaks.org/wiki/British_Telecom_Phorm_Page_Sense_External_Validation_report`

[4]As Syverson points out, the anonymity set is insufficient to analyze complete sender/receiver anonymity as a thorough analysis with realistic attacker strategies is appropriate [152]. However, we believe that the anonymity set is a tangible metric for evaluating topological anonymity that we aim to achieve in this chapter, and we leave it as future work to address various attacker strategies.

cannot link the sender and receiver if either *sender anonymity* or *receiver anonymity* is achieved. Since the design of a full anonymous communication system that can defend against timing attacks and conceal unique platform characteristics is beyond the scope of this chapter, we focus on concealing an endhost's network identifier and location in the network topology (which we call "topological anonymity"), which is an important step towards improving sender/receiver anonymity. For simplicity in expression, we also abbreviate "topological anonymity" simply with "anonymity" in the chapter.

Session unlinkability. Session unlinkability prevents an attacker from linking a user's activities over time. We want to ensure that given two packets from two different sessions, an attacker cannot determine whether these packets are associated with the same sender (or receiver).

Location privacy. Location privacy is achieved when a user conceals her *geographical* location so that an attacker cannot track her whereabouts.

Privacy levels. We want to provide different levels of privacy to endhosts under end-server attacks in case they are willing to trade privacy for improved performance [91].

In this chapter, we consider confidentiality of the packet payload to be orthogonal to the scope of our work as data confidentiality can be achieved using end-to-end encryption. Also, privacy leakage from higher layer protocols/payload is outside the scope of this chapter as such an issue can be alleviated by existing tools such as Privoxy.[5]

6.1.2 Desired Performance Properties

While providing an intermediate level of anonymity, we want to assure that the anonymity protection introduces marginal overhead. Following are the desired performance properties:

Low path stretch. We define path stretch as the increase in the number of AD hops normalized over the original (or non-anonymity) path length. Since the latency increases as the number of intermediate hops increase on the path, it is desirable to minimize path stretch.

Low performance overhead. We want to minimize cryptographic overhead, especially asymmetric operations and packet decryption and re-encryption at each hop.

Minimal state. To avoid the state explosion problem, we want to keep minimal or no per-flow state to reduce the attack surface and increase scalability.

[5] `http://www.privoxy.org/`

6.1.3 Assumptions

We assume that an end-user trusts her first-hop AD in the sense that the first-hop AD keeps its customers' information private and correctly performs anonymous forwarding protocols. This is aligned with the trust relationship in today's Internet since end-users place more trust on topologically closer ADs and generally have more control over the choice of their first-hop ADs than over the other ADs on a routing path. In case end-users do not trust their first-hop ADs and have no options to pick their own ADs, they may use anonymity systems such as encrypted tunnel IPsec, Tor [47], or anti-censorship systems [27].

We envision that ADs can control the amount of bandwidth allocated for anonymous communication, thus limiting the misuse of anonymous protocols, e.g., for sending untraceable attack traffic. We also assume that routers in ADs support packet-carried forwarding states.

6.1.4 Threat Model

An adversary's goal is to break the desired privacy properties described in Section 6.1.1 to discover the identity or location of a sender or a receiver of a given packet. More specifically, we focus on topology attacks where an attacker attempts to de-anonymize the sender (or receiver) using topological location information in a given AD-level topology, and leave it as future work to defend against timing correlation attacks [121, 60, 74, 31].

We consider a relaxed threat model with respect to the attacker's capability: the attacker can compromise any AD except the first-hop AD where the victim endhost resides. Under this model, our primary attack case is an end-server attack where a malicious server analyzes traffic to it or initiate communication with others. We also consider an in-network attack where a malicious AD beyond the first-hop of the victim endhost leverages its cryptographic keys to perform deep packet investigation or actively manipulate (e.g., inject, delete, delay, and replay) packets. Malicious servers and ADs can collude to share their knowledge base.

6.2 Overview: LAP

LAP is a lightweight protocol to facilitate real-time, bidirectional anonymous communication. In this section we first give a high-level overview of LAP, and explain how endhosts establish an encrypted path (e-path) and how ADs forward packets along the e-path to achieve an intermediate level of anonymity.

The core observation of this work is that encrypting path information (i.e., concealing forwarding information in the packet header) improves topological anonymity against an adversary in the LAP-setting since the adversary cannot retrieve the sender's (or receiver's) origin address from the packet. Moreover, extending an encrypted path to a benign AD increases the topological anonymity, simply because there are more potential origins whose paths could route through the AD. Extending an e-path beyond one hop is desired because one-hop encryption offers insufficient topological anonymity, as we demonstrate in the preliminary analysis (Section 6.4). We also discuss in Section 6.6 the level of anonymity when the adversary appears at different places on the e-path.

Background: Network setting. We consider a network consisting of Autonomous Domains (ADs) as the basic principal of inter-domain routing; each of these ADs has a set of interfaces, each with a unique ID, that can connect to neighboring ADs. ADs agree on an inter-domain routing protocol Θ, e.g., the Border Gateway Protocol (BGP). Upon receiving a packet destined to $dest$, an AD$_x$ evaluates $\Theta_x(dest)$ to determine the next hop of the packet.

Each AD maintains a master secret key, perhaps stored in a secure offline server, and derives short-term secret keys, each associated with a certain time period, from the master key. The actual encryption and authentication keys are derived from the short-term key and a nonce specified by the sender. We assume every gateway router in an AD has a copy of the short-term keys and knows how to process and route LAP packets within the AD.

LAP overview. At a high level, LAP has two phases, as shown in Figures 6.2 and 6.3. Suppose Alice wants to communicate anonymously with Bob without revealing her identity and precise location.

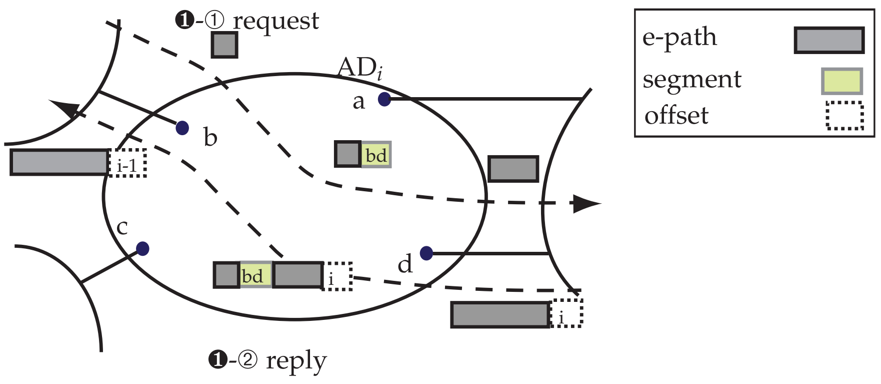


Figure 6.2: Operations within an AD. Step ❶-①: Upon receiving a request packet, an AD encodes its ingress (b) and egress (d) interfaces, extends the e-path in the packet, and forwards the packet (e.g., through interface d in this figure). Step ❶-②: An AD retrieves the interfaces from the e-path in the reply packet and forwards it (e.g., to interface b).

Phase ❶ Establishing e-paths: This phase enables Alice to obtain an e-path—a bi-directional routing path consisting of encrypted forwarding decisions by intermediate ADs on the path.

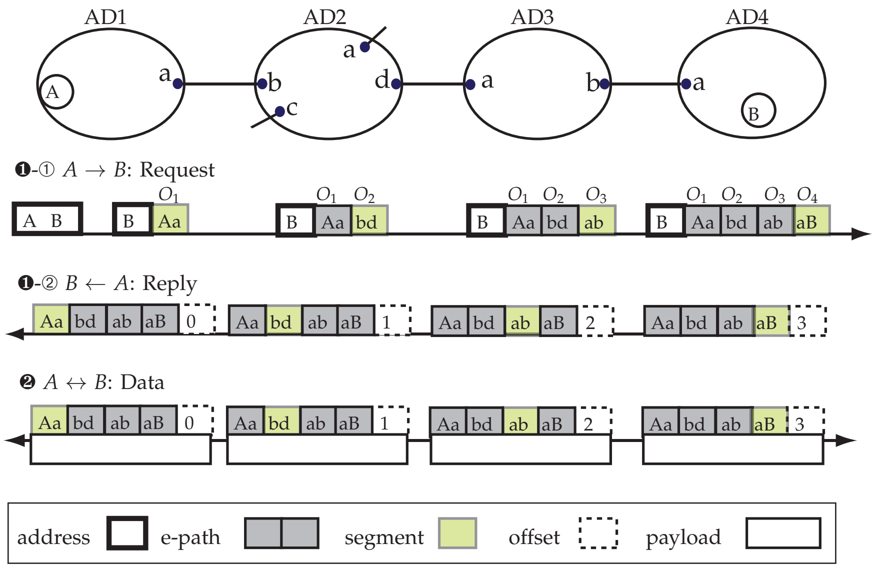


Figure 6.3: Operations between ADs. Step ❶-①: *A* sends a `request` to *B*, which is routed by *B*'s address. Step ❶-②: *B* replies e-path to *A* along the reverse path. An AD locates its segment by the offset pointer. Phase ❷: *A* and *B* send data to each other along the e-path.

- **Step ❶-① Request.** To set up an e-path as shown in Figure 6.3, Alice creates a `request` packet to reach Bob. When her `request` packet reaches a gateway router inside AD_1, it creates a segment which contains Alice's address along with the egress interface, encrypts the segment to anonymize Alice's origin address, and forwards the encrypted segment (O_1) to AD_2. Upon receiving the `request`, as shown in Figure 6.2, AD_2 encrypts its own forwarding decision in O_2 (i.e., the `request` packet from ingress interface *b* is forwarded to egress interface *d* to reach Bob), appends O_2 to the `request` packet, and forwards it to the next AD. This process continues until the `request` reaches AD_4, where Bob resides. Note that encryption and authentication of O_i use secret keys that are only known to AD_i so that only AD_i can later decrypt and verify O_i.

- **Step ❶-② Reply.** The resulting e-path enables Bob to send packets to Alice without knowing her origin address, because the e-path encodes the forwarding decisions made by ADs on the routing path. We leverage *packet-carried forwarding state*, where the network forwards packets solely based on the state contained in the header (i.e., e-path). More specifically, Bob retrieves the e-path from the `request` and puts the e-path in the header of a `reply` packet, which is a special type of `data` packet without payload. As shown in Figures 6.2 and 6.3, upon receiving the `reply`, AD_3 decrypts the segment O_3 that it encrypted during Step ❶-①, retrieves the egress interface *a*, and forwards the `reply` to the next hop. This process continues until the `reply` reaches the intended endhost Alice. If an AD fails to correctly decrypt or verify the segment, the `reply` is dropped.

Phase ❷ Forwarding: When Alice obtains the e-path from the `reply` packet, she can start sending `data` packets anonymously along this e-path using packet-carried forwarding state, as described above.

With LAP, Alice achieves sender topological anonymity and location privacy with respect to a LAP-setting adversary (e.g., Bob), because only her local AD knows her identity and address. In the following sections, we describe LAP in detail, and validate it using a real Internet topology. We also address the challenges of instantiating LAP in the current IP network and future Internet architectures.

6.3 LAP: Lightweight Anonymity and Privacy

In this section, we describe in detail how e-paths are constructed, and present additional mechanisms to achieve receiver anonymity and controllable privacy. We start with the packet header formats.

6.3.1 LAP Packet Header Format

Figure 6.4 illustrates the format of a LAP packet header. The header contains a 8-bit TYPE field to distinguish `request`, `reply`, forward `data` (from Alice to Bob), and backward `data` (from Bob to Alice) packets (six bits of the TYPE field are reserved for future extensions). The header also contains a 32-bit NONCE field to assist session unlinkability.

`Request.` A `request` packet indicates Alice's intent to anonymously communicate with Bob. To initiate a request, Alice specifies Bob's address in a 32-bit DESTIP field and her desired privacy/performance tradeoff, expressed in a 8-bit HOP-TO-ENCRYPT (HTE) field (to be discussed in Section 6.3.3). As the request travels through ADs until it reaches Bob, each intermediate AD appends its own encrypted path segment to the E-PATH field (to be described later).

`Reply.` A `reply/data` header contains no IP address since `reply/data` packets can be forwarded using the bi-directional e-path that is copied from the corresponding `request` packet. The header also contains a LENGTH field to indicate the size of the packet, and an OFFSET field to indicate the appropriate segment from the E-PATH field that the receiving AD can decrypt. AD_i adjusts the OFFSET field based on the direction of the packet (e.g., for `reply`, OFFSET is decreased by 1).

`Segments in e-path.` The E-PATH field comprises a sequence of segments, each of which is 128 bits by default. As shown in Figure 6.4, an AD creates each segment consisting of INGRESS and EGRESS interfaces, size of the segment, RESERVED to store additional information (e.g., source AD can store the source IP address which does not fit in the INGRESS field), and MAC to store the Message Authentication Code over

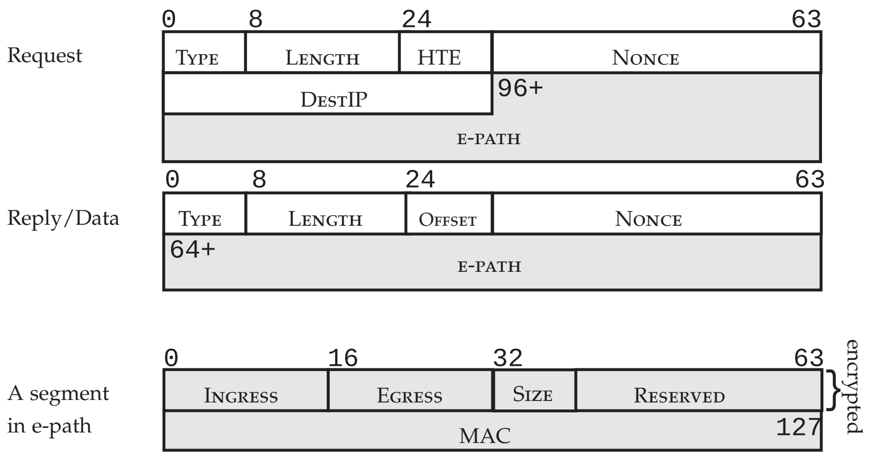


Figure 6.4: LAP packet header formats. In a segment, the first 64 bits are encrypted, and the Reserved field can be used to store additional information of an AD.

all segments in the e-path field (including its own). Note that LAP can support variable-size segments in multiples of 128 bits (and thus a Size field is needed in a segment) to defend against size-based passive traffic analysis, as discussed in Section 6.3.5.

6.3.2 LAP Protocol Description

We now describe Phases ❶ and ❷ in detail.

Encrypted path establishment. To construct an e-path, Alice sends a `request` to Bob (Step ❶-①), and by default, LAP requires each AD to append its encrypted routing decision to the received request packet.

Suppose Alice resides in AD$_1$ and Bob resides in AD$_n$, and the request packet moves along a path AD$_1$, AD$_2$, . . ., AD$_n$. As shown in Figure 6.3, AD$_i$ generates a segment O_i, which contains the encrypted ingress and egress interfaces for bi-directional forwarding, and appends to the packet. As a result, a resulting e-path $O_{A,B}$ consisting of $\{O_1, \ldots, O_n\}$ is constructed as follows: let $O_0 = \emptyset$. For $i = 1 \cdots n$,

$$\chi_i = Enc_{k_i^e}(M_i),$$
$$O_i = \chi_i \| MAC_{k_i^s}(\chi_i \| O_{i-1}) \tag{6.1}$$

where M_i contains an AD's routing decision (i.e., the ingress and egress interfaces), $Enc_k(m)$ means encrypting m using key k, and $MAC_k(m)$ is the Message Authentication Code of m using k. k_i^e and k_i^s are symmetric keys derived from the nonce and the AD$_i$'s current short-term key, known only to AD$_i$.

We include the previous segment in the MAC computation to enforce the routing decision while preventing attackers from crafting an arbitrary path. Without MACs, an adversary can easily find a ciphertext decrypted to some meaningful egress/ingress interfaces. Simply adding a regular MAC is insufficient because an adversary may be able to craft an invalid path by combining segments obtained from two separate requests. Hence, in LAP, we use layered MACs to prevent arbitrary combinations of segments.

AD$_i$ appends O_i to the E-PATH field of the `request`, and forwards it to AD$_{i+1}$ (via egress interface) until the `request` reaches Bob.

Packet-Carried Forwarding State. For successful packet forwarding using packet-carried state, endhosts copy the E-PATH field from the preceding packet. For example, upon receiving the `request`, Bob copies the E-PATH field to the `reply` packet. Similarly, when Alice receives the `reply`, she copies the E-PATH to the `data` packet, and Alice and Bob copy the E-PATH for succeeding `data` packets.

Using the e-path in a `reply` packet (Step ❶-②) and a `data` packet (Phase ❷), ADs can forward the `reply`/`data` packet along the encrypted path bi-directionally without actually knowing Alice's or Bob's address. Suppose a `reply` packet enters an AD$_i$ from interface d, as shown in Figure 6.2. The AD proceeds as follows:

1. *Retrieve forwarding decision:* It first locates its segment O_i based on OFFSET and TYPE (which encodes the direction of forwarding) in the header. It then decrypts χ_i to recover the ingress interface ig, egress interface eg.

2. *Verification:* O_i is valid if the following conditions hold: i) for a forward packet (e.g., `data`), $d = ig$; for a backward packet (e.g., `reply`, `data`), $d = eg$, and ii) MAC verification succeeds (i.e., the AD re-computes the MAC using its current secret key and the information embedded in the header, and checks if the resulting MAC matches the one included in O_i.)

3. *Forwarding:* If this segment is valid, the AD determines the exiting interface and adjusts the offset. In our example, since this is a backward packet, the exiting interface is ig and the offset should be decreased by 1. The AD then forwards the packet to the exiting interface.

Since ADs rotate their short-term keys periodically (e.g., every hour) for security, Alice may have to renew or request a new e-path if any key for decrypting or verifying the e-path expires during her session. LAP can support efficient renewal by embedding updated e-path in data packets.

Session unlinkability. Alice can request a new e-path (by specifying a different nonce) for every new session to achieve session unlinkability. Also, the encryption algorithm should be secure against chosen-plaintext attacks such that encrypting the same plaintext twice would result in two different ciphertexts with high probability. For example, one can use AES in CTR mode. The initialization vector (IV) in CTR mode can be derived from the nonce and the previous O_i to avoid allocating extra space for storing IV in the packet. Since a different nonce or routing path would result in a new e-path, an attacker has a low success rate in correlating two separate sessions based on an e-path.

6.3.3 Controllable Privacy Levels

Encrypting every AD hop in LAP increases the packet header size and computational overhead, and may reduce the flexibility in routing (e.g., in the case of multipaths, the sender cannot make an informed decision in path selection without knowing which ADs are on the path.) Although LAP provides negligible computational overhead on routers (see Section 6.7) and we anticipate that routers will be improved to support larger packets, users may still want to trade privacy for improved performance.

LAP provides options for endhosts to control the length of e-paths, which results in differentiated privacy disclosure. The intuition is that the degree of anonymity and privacy (in terms of the size of an anonymity set) increases with the length of an e-path (in terms of the number of AD-hops). More specifically, Alice specifies the desired length of the e-path in a Hop-to-Encrypt (HTE) field in the `request` packet. Each AD checks the HTE field before updating the e-path, and if HTE ≥ 1, the AD updates the `request` packet as usual and decreases the HTE field by 1. If HTE reaches zero before reaching Bob, the intermediate AD returns the e-path to Alice on a `reply` packet. Similarly, if Bob receives the packet with HTE ≥ 0, Bob returns the e-path to Alice on a `reply` packet. Note that to use such partially encrypted paths, packets have to contain an extra field storing the destination's address (which, however, can be in plaintext, as receiver anonymity is provided using rendezvous points, as will be explained in Section 6.3.4). During the forwarding phase, the AD at the end of the e-path converts `data` packets between the LAP- and regular-mode. For example, in BGP routing, the AD encapsulates the e-path in a normal IP packet and sets the source address to be its own address and the destination address to Bob's.

6.3.4 Path Publishing for Receiver Anonymity

We have shown that Alice can achieve sender anonymity and location privacy by constructing an e-path to Bob (i.e., only Alice's first-hop AD knows her identity and location). However, sometimes Bob may want privacy protection as well. For example, a user running a controversial website (e.g., WikiLeaks) would prefer to hide his location and permanent identity to prevent tracking or avoid censorship. However, since a receiver is unaware of who a sender might be in advance, the challenges become (1) how the receiver constructs an e-path for any potential sender and (2) how a sender looks up the receiver's e-path without knowing his permanent identity.

At a high level, to achieve receiver anonymity, Alice and Bob each initiate an e-path to a *rendezvous point* so that only the local ADs know the identity of endhosts. Such an indirection technique is commonly used in anonymity systems [47]. To address the second challenge, Bob publishes his e-path associated with his pseudonym on a publicly-accessible *Path Server*. As a result, a sender knowing Bob's pseudonym (e.g., via out-of-band communication) can retrieve Bob's e-path from the Path Server and reach Bob through the rendezvous AD. In theory, any AD in the Internet could be a rendezvous point or host a Path Server. To minimize the path stretch and communication overhead, in practice, tier-1 ADs are a reasonable choice of rendezvous ADs and Path Server administrators, because most of the Internet traffic goes through tier-1 ADs.

6.3.5 Padding Against Size-Based Traffic Analysis

If we use fixed-size segments, an attacker can determine the distance (in terms of AD hops) to a sender based on the size of the header. Hence, LAP allows ADs to pad segments (variable-size segments) to enhance topological anonymity. As mentioned in Section 6.3.1, the size of each variable-size segment is in multiples of 128 bits. For proper decryption and adjustment of the offset, each AD needs to know the size of its own segment. Hence, to allow proper operations on both forward and backward packets, an AD using a variable-size segment encodes the size in both the first and last 128-bit blocks in the SIZE field as follows: AD$_i$ (1) creates the first 128-bit block O_i using symmetric key k_i as described in Section 6.3.2; and (2) copies the same INGRESS, EGRESS, and SIZE to the last 128-bit block of its segment, and creates the MAC over the entire segment using another symmetric key k_i'. In this manner, the first 128-bit block looks different from the last 128-bit block. With this process, the AD can recover the length of its own segment from either the first or the last 128 bits of the segment, and adjust the offset properly. For (1), note that since an AD does not know the size of the previous segment, it computes a MAC over the last 128 bits of the previous segment.

With these variable-size segments, an attacker can only obtain an upper bound on the distance to the sender, which is the size of the e-path in bits divided by 128. The optimal way of padding results in an e-path of $128 \cdot l$ bits, where l is the distance of the farthest potential sender in AD hops.

6.4 Preliminary Analysis

In this section, we illustrate that the current Internet provides minimal anonymity, and demonstrate how LAP can increase the level of anonymity with a real Internet topology.

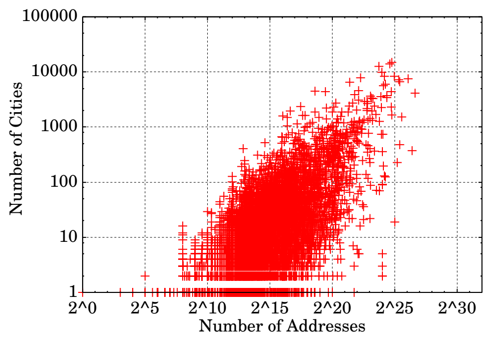


Figure 6.5: As the number of possible IP addresses increases, so does the number of potential cities.

6.4.1 Anonymity and Privacy in the Current Internet

Anonymity in the Internet is hindered by long lasting endhost identifiers, namely IP addresses. From a network layer's perspective, IP addresses identify both the source and the destination of the traffic. Hence, by snooping on traffic flows, malicious nodes can easily determine which endhosts are communicating with each other and link different sessions to the same endhosts. While public servers prefer long-lasting IP addresses for availability, current Internet protocols and ISP policies generally assign IP addresses that last on the order of days[6] to clients who have no desire to run public servers. Typically, these IP addresses (from the ISP's allocated address space) change only when the DHCP lease time expires. While NAT boxes can provide an anonymity set greater than one, devices behind them are usually both small in number and in the same geographical area, thus providing extremely limited privacy guarantees. In the cellular realm, the situation is better since providers' NATs can mask a wider range of clients [165]. Ideally, privacy solutions should be available in all domains that easily allow endhosts to retain anonymity at the network level.

Consequently, while the current Internet intrinsically provides a certain level of anonymity based on dynamic addressing techniques (e.g., DHCP and NAT), the degree of anonymity is constrained by the size of the IP prefixes. More specifically, we estimate the anonymity set size by analyzing the announced prefix sizes and the number of subscribers of six main ISPs in the U.S., as Table 6.1 summarizes. We group the prefixes (extracted from the RouteViews dataset [8]) into ISPs using AS description from the CIDR report[7]. Assuming that subscribers are uniformly distributed in an ISP's address space, the size of an anonymity set can be as low as $2^{4.7} \simeq 26$.

Similar studies have shown that hiding behind a prefix provides insufficient anonymity [136]. Although aggregating prefixes associated with the same location may increase the size of the anonymity set

[6]DHCP Best Practices. http://technet.microsoft.com/en-us/library/cc780311(WS.10).aspx
[7]http://www.cidr-report.org/as2.0/

Table 6.1: Anonymity set size of US top ISPs.

ISP	Address Space (Entropy)	Announcing Prefix	Subscriber [103] (Entropy)	Subscriber Entropy/Prefix		
				Ave.	Min	Max
Comcast	70,374,912 (26.1)	865	17,406,000 (24.1)	19.5	6.0	22.0
Time Warner	27,556,352 (24.7)	2,158	9,992,000 (23.3)	14.4	6.5	17.5
Cox	11,971,584 (23.5)	1,507	4,400,000 (22.1)	18.8	6.6	19.6
ATT	114,544,128 (26.8)	6,127	16,485,000 (24.0)	18.3	5.2	21.2
Verizon	84,403,200 (26.3)	4,376	8,490,000 (23.0)	15.5	4.7	19.7
Quest	84,403,200 (24.0)	899	2,965,000 (21.5)	16.2	5.5	18.5

(but not location privacy), the flexibility of route management within an ISP may diminish. Also, users have no control over their level of anonymity.

We also investigate location privacy in the current Internet. We use the Maxmind GeoIP locationing tool to estimate an endhost's current city[8] based on its IP address and quantify the location-privacy level based on the number of cities the endhost may reside in. Figure 6.5 shows the relationship between the number of cities and the anonymity set size: the level of location privacy can be increased by increasing the number of possible IP addresses.

6.4.2 Anonymity in LAP

In LAP, users can improve their anonymity set size by extending the length of their e-paths.

To show the effectiveness of LAP path encryption, we evaluate anonymity in LAP using traceroute data from iPlane's measurements and routing data from RouteViews [8]. The iPlane dataset contains traceroute data between 197 sources and about 13 thousand destinations. We eliminate 28 sources with incomplete logs and choose 1,000 destinations for each source. For each pair of source and destination, we calculate the size of the source anonymity set with respect to the destination based on the Internet topology and the assigned address space extracted from the RouteViews dataset. According to Figure 6.6, which illustrates the CDF (cumulative distribution function) of the number of addresses, the increase in the number of encrypted hops increases the anonymity set.

6.5 LAP Instantiation

In this section, we discuss how LAP can be accommodated in the current IP network running BGP. We then discuss the potential benefits of tailoring LAP to two future Internet architectures: SCION [178] and MobilityFirst [2].

[8]Maxmind determines city names based on the Geographic Names Data Base. http://www.maxmind.com/

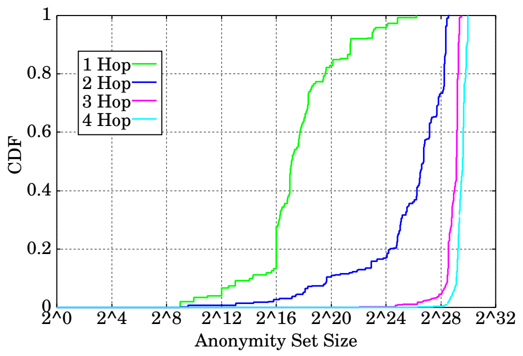


Figure 6.6: Comparison of anonymity set size based on the number of encrypted hops. As the number of encrypted AD hops increases, the anonymity set size increases. For the case of four encrypted hops, almost all origins enjoy an anonymity set size of over 2^{28} hosts.

6.5.1 LAP in the Current Internet

In this section, we delineate how LAP can be incrementally deployed in the current IP network. We consider both LAP-enabled ADs and legacy ADs that do not support LAP. In such heterogeneous networks, one main challenge is to enable a LAP-enabled AD to discover and build virtual channels to nearby LAP-enabled ADs. For this integration, we assume that the IP header contains a *LAP-flag* bit that is set if an IP packet encapsulates a LAP packet.[9]

A legacy AD is agnostic to the encapsulated LAP packet and routes IP packets based on the destination IP as specified in the IP packet header. A LAP-enabled AD, on the other hand, installs dedicated LAP routers where each of them has a publicly-accessible address, and configures every gateway router to route LAP packets (whose LAP-flag is set) to the nearest LAP router. Figure 6.7 illustrates a scenario where AD_1 and AD_3 are legacy ADs, and AD_2 and AD_4 are LAP-enabled ADs. X and Y represent the LAP routers in AD_2 and AD_4, respectively.

When Alice (whose IP address is A) wants to diffuse her topological location for her communication with Bob (whose IP address is B), she installs a LAP application proxy on her machine. To obtain an e-path, this proxy prepares a LAP `request` packet and encapsulates it in an IP packet. Then, this IP packet is initiated with $srcIP = A$ and $destIP = B$.

[9]Several potential approaches exist to add LAP to the current IP header. One approach would be to add a LAP IP options field, however, this would constrain the length of the LAP header and possibly also slow down packet processing at legacy routers. Another approach would be to use a bit in the current IP header to indicate presence of a LAP header. We could use bit 0 of the 3-bit FLAGS field, which is currently unused. Another potential use could be a bit within the TYPE OF SERVICE OR DIFFERENTIATED SERVICE byte, since the PRECEDENCE or the ECN bits are rarely used. Yet another approach would be to set the PROTOCOL field to indicate that the next header is a LAP protocol header. In the two latter cases, the LAP header could be placed between the IP and TCP or UDP headers.

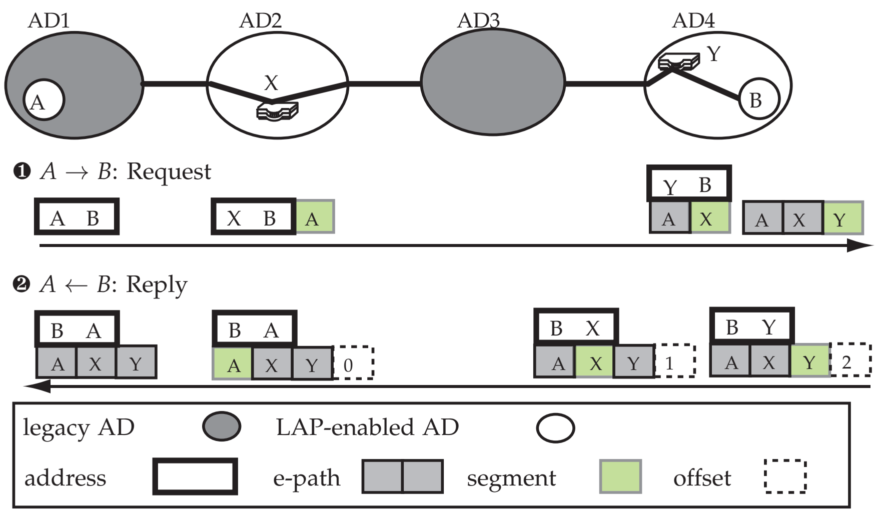


Figure 6.7: Incremental deployment of LAP in the current Internet.

Encrypted path establishment. The `request` packet sets up an anonymous return path by which Bob can reach Alice without knowing her IP address. When a gateway in the LAP-enabled AD_2 receives a LAP-flagged `request` packet, it routes the packet to the dedicated LAP router X. X then encrypts the $srcIP$ to generate its e-path segment O_2 and appends O_2 to the encapsulated LAP packet. X also updates the $srcIP = X$ in the IP header but $destIP$ remains the same. Similarly, AD_4 process the packet in the same way. When Bob, receives a packet whose $srcIP = Y$ and $destIP = B$, he sends a `reply` packet with $srcIP = B$ and $destIP = Y$. We assume that the LAP-flag and LAP header are preserved in the `reply` packet. When router Y receives the `reply`, it verifies O_4, extracts the IP of the previous LAP router (i.e., X) from O_4, and updates the destination address to be X. Similarly, router X retrieves A from O_2 and updates $destIP = A$.

Forwarding. Alice obtains an e-path from the `reply` packet. To send a data packet to Bob, Alice prepares a LAP `data` packet that contains the e-path and encapsulates it in an IP packet whose $srcIP = \emptyset$ and $destIP = B$. Upon receiving a LAP data packet, Bob returns data packets using the embedded e-path, as described above. Note that ADs can distinguish forward and return `data` packets based on the TYPE field and adjust the OFFSET correctly.

Asymmetric paths. Another advantage of LAP integrated with the current Internet is that it can support asymmetric inter-domain paths, which may exist in BGP due to routing policies, because in this instantiation LAP path is defined by a list of IP addresses instead of interfaces.

6.5.2 Integrating LAP into SCION

In this section, we show that LAP can be seamlessly integrated into SCION [178], which is reviewed in Section 2.2. LAP only requires an overhead for path establishment and encryption/decryption of packet-carried forwarding information, because packet-carried forwarding state and encrypted path publishing/downloading can be embedded into the existing SCION framework.

We begin with a strawman proposal in which the attacker can link two e-paths to each other. Learning from this strawman proposal, we present an integration that preserves session unlinkability.

Strawman proposal. Since SCION constructs (unencrypted) Packet-Carried Forwarding State (PCFS) as PCBs travel from the ISD Core to endpoint ADs, a straightforward integration suggests itself: when constructing PCBs, intermediate ADs could in addition encrypt SCION's PCFS, and as a result each endpoint AD could obtain a set of e-paths to reach the ISD Core without explicit requests. However, this approach violates the session unlinkability property because packets carrying e-paths with a common prefix must originate from the same topological region.

Path encryption requests. The violation of session unlinkability in the strawman proposal is due to the dependency between e-paths. Thus, a better approach would be sending separate `requests` for each individual e-path, as our LAP protocol specifies. Essentially, given a routing protocol, LAP forwards `request` packets using the routing protocol and forwards `reply` and `data` packets using the established e-paths. Despite its simple setup, the actual challenge here is to ensure the routing protocol itself (SCION in this case) does not leak information that helps identify requestors or link packets.

In the following, we first investigate two privacy attacks that exploit SCION's routing information when forwarding LAP `requests` using SCION. To mitigate these attacks, we then propose minimal modifications to SCION routing on the premise of not jeopardizing SCION's guaranteed properties, such as controllability, isolation, and scalability.

- *Privacy Leak 1: Unconcealed path information in the requests.* Since SCION routing operates on PCFS, forwarding a packet in SCION requires putting a full path (i.e., timestamped PCFS by which packets can be routed from the source AD to the destination AD) in the packet. As a result, an attacker can easily link two `requests` (and therefore two different sessions) by inspecting the unencrypted path information in the `request` packets.

- *Privacy Leak 2: Non-uniform path distribution.* The second attack, which is subtler compared to the first attack, leverages the fact that, given a routing path, the probability space of the senders is likely to

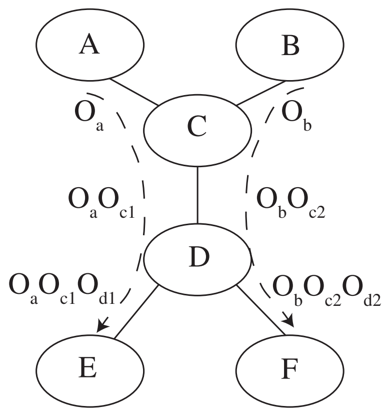


Figure 6.8: In this example, AD D selectively propagates $O_aO_{c1}O_{d1}$ to AD E and $O_aO_{c2}O_{d2}$ to AD F. Due to this non-uniform path propagation, AD C can deduce that requests carrying O_{c1} are likely from the same sender.

be non-uniform. Formally, when an AD$_i$ sees a request and the corresponding unencrypted opaque field O_i, it is unlikely that $Pr[s_x|O_i] = Pr[s_y|O_i]$ for any s_x, s_y, O_i, where s_x and s_y are two senders. Figure 6.8 illustrates an example in this attack category. Since AD D selectively propagates the PCBs to different customer ADs, AD C can deduce that requests carrying O_{c1} are likely from the same sender.

Fixing Privacy Leak 1 is easy. Upon receiving a `request`, every intermediate AD removes the (unencrypted) previous hop information to erase the trace. Removing previous hops in the unencrypted path information does not affect the normal operations in LAP-enhanced SCION, as the following packets (i.e., the `reply` and `data` packets) in the same session are sent on the newly-established e-path and do not require knowing the path information in the cleartext.

To fix Privacy Leak 2, one could require ADs to propagate every PCB to every child AD, such that every possible route is learned. However, this approach is expensive. A more efficient fix called *opaque field swapping* is as follows. An intermediate AD always replaces the remaining part of the unencrypted packet-carried forwarding state in a request packet with a fixed packet-carried forwarding state for the same route, such that an attacker sitting beyond this hop cannot gain information by inspecting the remaining part of the path. This requires an AD to store at least one fixed PCFS for each route it has advertised recently. While this approach seems to overly expand the power of en route ADs, we note that SCION intentionally uses a lightweight authentication mechanism for high efficiency and thus does not prevent en route ADs from stealthily diverting packets.

In sum, SCION ADs route packets using (unencrypted) packet-carried forwarding state and verify the forwarding information using MACs. Hence, running LAP with SCION requires adding symmetric encryption/decryption functions to routers. In SCION, a source obtains a set of paths to reach the destination for source-selection routing. Hence, Alice embeds a `request` packet inside a SCION packet by specifying one of the (unencrypted) paths for an e-path construction. Upon receiving this packet, an intermediate AD (AD$_i$) appends its O_i, removes the unencrypted counterpart to erase the trace, and replaces the remaining unencrypted forwarding state with the fix one for the same route.

The `reply` and `data` forwarding can be done as described in the LAP protocol section (Section 6.3.2).

Path Server and rendezvous points in ISD Cores. The design of SCION requires a Path Server to store ADs' downstream paths, as an end-to-end path is constructed by splicing a source-to-core path with a core-to-destination (downstream) path. Similarly, LAP also requires a Path Server that stores encrypted paths to certain rendezvous points. Hence, SCION Path Servers can manage both SCION paths and LAP's encrypted paths. In this manner, an ISD Core becomes a default rendezvous point since all paths can traverse the ISD Core. Note that for the sake of efficiency (shorter paths), SCION may permit shortcuts that bypass an ISD Core by comparing and finding the intersection of the upstream and downstream paths. However, in LAP, finding such common intersections (common links or ADs) when the intersections are encrypted is fundamentally infeasible because an attacker could take the intersection finding algorithm as an oracle to decipher encrypted paths. Fortunately, the semi-encrypted paths (constructed by setting a small Hop-To-Encrypt value in the `request`) in LAP enable part of a path to be encrypted for a sufficient degree of privacy with the other half remaining unencrypted to enable shortcut construction.

6.5.3 Integrating LAP into MobilityFirst

To further illustrate the flexibility of LAP, we now describe how it can also be integrated into a mobility-centric future Internet architecture called *MobilityFirst* [2]. MobilityFirst retains a distributed routing control plane similar to that of BGP, while providing a clean separation of network "entities" and routable addresses. Privacy is a major concern for mobility-centric architectures since they allow humans, via devices they carry or drive, to be continuously connected to the broader Internet. Hence both control-plane reachability updates as well as content generated by these devices have the potential to breach privacy. Low-stretch privacy solutions that cleanly integrate with mobility-centric architectures can give end users privacy with minimal disruption. As with SCION, LAP naturally complements MobilityFirst and adds little overhead.

Background of MobilityFirst. MobilityFirst is a clean-slate Internet architecture designed to address challenges brought about by an increase in the number of mobile, wireless devices. At its core, MobilityFirst provides a mechanism to abstract network entities important to applications, and bind those abstractions into routable network addresses. Specifically, entities such as an individual laptop, a vehicle, a piece of content, or a group of people each obtain a *globally unique identifier*, or GUID, that the application uses for communication. When data destined for a GUID is received by a MobilityFirst router, the router will either attempt to directly route on the GUID or bind the GUID to a routable address via a massively distributed *global name resolution service*, or GNRS. All publicly available entities are responsible for ensuring that their GUID-to-network address mapping is up-to-date in the GNRS. The GNRS is accessible from all MobilityFirst routers and hence GUIDs can easily be re-bound deeper in the network if the destination's network address has changed. In addition to separating naming from addressing, MobilityFirst heavily utilizes in-network storage and hop-by-hop transfer of large data chunks to react to network and host mobility.

Path encryption requests. MobilityFirst's low-level routing plane is similar to that of BGP, with the exception of IP prefix announcements. Since the GNRS handles the "who is in what network" question, MobilityFirst routing simply needs to exchange AD-level reachability information. A LAP path encryption request will occur after a MobilityFirst router (e.g., the border router of the source AD) queries the destination GUID for a destination network address. The destination network address can be used as the destination of a path encryption request. This process, as described in Section 6.3.2, can then proceed as it would with BGP.

Path Server and rendezvous points. The GNRS is responsible for binding GUIDs to routable addresses, and hence is a perfect match for the LAP Path Server. Using LAP, the GNRS will bind a GUID (which may be a pseudonym) to an e-path leading to a rendezvous point. Therefore, a router wishing to route towards a destination GUID will make a GNRS query and either get back the destination network address or an e-path leading to a rendezvous point. MobilityFirst networks, however, do not have a strict hierarchy, and hence choosing a rendezvous point is less intuitive. However, since the GNRS is capable of handling multihomed GUIDs, multiple rendezvous points can be uploaded and bound to the same GUID. If the destination also provides hints, such as "use encrypted path 3 if in North America", this can alleviate stretch problems at the expense of some decrease in location privacy.

Handling mobility. In order to dynamically respond to mobility and disconnection deep within the network, the destination GUID is always available as the authoritative header on a piece of data. Routers detecting a problem with a destination network address can always query the GNRS and re-bind the

GUID to a new destination address. LAP integration does not change this, as the destination GUID can always be re-bound to a new e-path obtained from the GNRS.

6.6 Security Analysis

We analyze how LAP conceals endhosts' topological locations for an intermediate level of anonymity and achieves session unlinkability. We also describe how LAP defends against attacks.

6.6.1 Sender/Receiver Anonymity Analysis

In this analysis, we consider a scenario where Alice and Bob communicate with each other along an AD path AD_1, AD_2, $\cdots$, AD_n and quantitatively analyze the degree of anonymity with respect to an adversary, *adv*, at various vantage points on the path.

We compare LAP with three related anonymous systems: Tor [47], Tor Instead of IP [108], and AHP [136]. We show that LAP provides a competitive degree of anonymity compared to low-latency anonymity systems in the presence of LAP-setting adversaries. Also, LAP guarantees much stronger anonymity properties compared to AHP, which provides a limited level of protection due to a small anonymity set and does not support receiver anonymity.

Notation. We denote $\mathbb{A}_s^{adv}(x)$ as the sender anonymity set of user x with respect to adversary adv. The receiver anonymity set $\mathbb{A}_r^{adv}$ is defined similarly. Let N be the total number of Internet users. Thus, N is the maximum size of an anonymity set. N_t is the number of Tor users and $N_t \leq N$. In practice, $N_t \ll N$ because N_t is between 10^5 – 10^6[10] while N is on the order of 10^9[11].

Assumptions. As mentioned in Section 6.1.1, a sender can achieve stronger anonymity if its identity is hidden in a larger anonymity set. For the analysis, we assume equiprobability for subjects in an anonymity set. That is, an adversary can determine who may have sent or received a packet within a given anonymity set but cannot tell whether one is more likely to send/receive than the others in the same set. We consider a LAP-setting adversary, who can leverage topological information but not timing information and cannot compromise the first-hop AD of a victim. An adversary with the knowledge of the AD-level topology can narrow down the anonymity set of a packet based, for example, on the length of the packet header and the packet's incoming interface. For this analysis, we assume full deployment of LAP, Tor Instead of IP, and AHP.

[10]Tor Metrics Portal: Users. `https://metrics.torproject.org/users.html`
[11]Internet World Stats. `http://www.internetworldstats.com/`

Table 6.2: Comparison of sender and receiver anonymity, represented by the pair of ($|\mathbb{A}_s^{adv}(Alice)|$, $|\mathbb{A}_r^{adv}(Bob)|$). Assume full deployment of LAP, Tor instead of IP, and AHP. $|AD_x|$ is the number of clients in AD_x.

	Adversary *adv*		LAP	Tor [47]	Tor instead of IP [108]	AHP [136]
LAP-setting	adv1	AD_n	$(N, \text{n/a})$	$(N_t, \text{n/a})$	$(N, \text{n/a})$	$(\leq \lvert AD_1\rvert, \text{n/a})$
	adv2	AD_i $(v < i < n)$	$(N, < N)$	(N_t, N_t)	$(N, < N)$	$(\leq \lvert AD_1\rvert, 1)$
	adv3	AD_v (or Tier 1)	$(\approx N, \approx N)$	(N_t, N_t)	(N, N)	$(\leq \lvert AD_1\rvert, 1)$
	adv4	AD_i $(1 < i < v)$	$(< N, N)$	(N_t, N_t)	$(< N, N)$	$(\leq \lvert AD_1\rvert, 1)$
	adv5	AD_1	(n/a , N)	(n/a, N_t)	(n/a, N)	(n/a, 1)
non-LAP-setting	adv6	AD_n	$(N, 1)$	$(N_t, 1)$	$(N, 1)$	$(\leq \lvert AD_1\rvert, 1)$
	adv7	AD_1	(1, N)	$(1, N_t)$	$(1, N)$	$(1, 1)$
	adv8	adv6+adv7	(1,1)	(1, 1)	(1,1)	$(1, 1)$

We summarize our analysis in Table 6.2, where the first column describes the adversary's location and the following columns present ($|\mathbb{A}_s^{adv}(Alice)|$, $|\mathbb{A}_r^{adv}(Bob)|$) for LAP, Tor, Tor instead of IP, and AHP. Below, we justify the table.

1) LAP: In this analysis, we consider LAP with full path encryption (Alice's e-path + Bob's e-path through a rendezvous AD AD_v in Tier 1) and optimal padding. Hence, a malicious AD can conclude that the sender (or receiver) must reside in an AD that is reachable from the incoming (or outgoing) interface. However, because of optimal padding, an attacker cannot obtain identifiable information from the size of the header.

In LAP, only the first- or last-hop AD knows the identity of the sender or receiver, respectively. Hence an adversary cannot link the sender and the receiver in LAP unless he controls both the first and the last ADs along the path (adv8 in Table 6.2), which is, however, outside our threat model. Moreover, the degree of anonymity increases with the length of the e-path. In other words, the farther away an attacker is from the user, the higher the degree of anonymity. For example, if Bob is an attacker (adv1 in Table 6.2), Alice's sender anonymity set is N, because Bob has no knowledge of the interface information, and every Internet user could be the sender from Bob's point of view. On the other hand, if Alice's first-hop AD is the attacker (adv7), her anonymity set is 1.

Generally, the degree of anonymity strictly increases as the attacker's position moves toward AD_v (adv3), because for each additional AD between Alice and the attacker, users in that AD are added to the anonymity set:

$$|\mathbb{A}_s^{AD_i}(A)| \geq |\mathbb{A}_s^{AD_j}(A)| + |AD_j|$$
$$\Rightarrow |\mathbb{A}_s^{AD_i}(A)| > |\mathbb{A}_s^{AD_j}(A)| \text{ if } v+1 \geq i > j$$

If the attacker is beyond AD_v (adv4), the anonymity set is $|\mathbb{A}_s^{AD_{v+1}}(A)|$ because the rendezvous AD is known. That is, $|\mathbb{A}_s^{AD_i}(A)| = |\mathbb{A}_s^{AD_{v+1}}(A)|$ if $i > v+1$. Therefore, when the attacker is on the path between Bob and AD_v (including Bob), Alice has the highest degree of anonymity, where any endhost in the network could be the sender (assuming that AD_v is reachable from all endhosts).

Finally, colluding ADs can easily share knowledge and correlate packets since LAP conceals neither packet content nor packet size. Thus, the resulting anonymity set is the intersection of those perceived by individual malicious ADs. Also, LAP provides no anonymity if both endpoint ADs collude.

2) Tor [47]: For the purpose of this analysis, we assume that Alice and Bob are Tor clients but do not serve as Tor relays. An attacker can learn a list of Tor relays from Tor directory servers. Hence Alice's first-hop AD (AD_1) can observe that she is sending packets. However, the second-hop AD (AD_2) cannot learn the origin of the packet because it cannot distinguish whether the Tor sender resides in AD_1, or the packet is relayed by other Tor servers and routed through AD_1. In general, if an attacker is an AD except AD_1, Alice is hidden within all active Tor users (N_t). The same analysis can be applied for receiver anonymity. Unlike LAP, Tor can prevent colluding ADs from linking Alice with Bob based on topological or packet information, because layered-encrypted packets look different at each AD. However, Tor is vulnerable to timing attacks performed by colluding ADs (e.g., adv 8).

3) Tor Instead of IP [108]: Recent proposals identify the importance of improving the default privacy level at the network layer. Instead of using Tor as an overlay, Liu et al. propose replacing IP with Tor. They assume that each AD runs a Tor server, and that packets travel from the sender to the Internet core (Tier 1) and then to the receiver similar to LAP rather than being routed via an indirect path. Tor instead of IP, however, allows zigzag paths in the core to improve anonymity. Hence, this scheme exhibits the same level of anonymity as LAP when an attacker is not at the core, but a slightly better anonymity when the core AD is malicious. However, in terms of performance, this scheme suffers from expensive path establishment and stateful communication similar to Tor.

4) AHP [136]: Raghavan et al. propose Address Hiding Protocol (AHP), in which an ISP shuffles its own address space and assigns a random IP to a sender. Trostle et al. present a similar approach to enhance sender's location privacy using Cryptographically Protected Prefixes (CPP) [157]. Both AHP and CPP achieve a level of sender privacy constrained by the available address block and geographical distribution of the sender's hosting ISP. For example, the sender anonymity in AHP is bound by the size of the first-hop AD (or ISP). Also, they do not offer receiver anonymity or location privacy.

6.6.2 Session Unlinkability

Session unlinkability can be achieved by requesting a new e-path for every new session. Furthermore, a sender can refresh paths more frequently or use more than one path simultaneously, thanks to the lightweight construction of an e-path. Hence, LAP does not require the same path to be reused for multiple TCP sessions. We show that LAP achieves session unlinkability by considering the knowledge of a malicious AD in the LAP-setting as follows. From a `request` packet, an AD knows an e-path to the sender, the size of e-path (which provides an upper bound on the AD-level distance to the sender), the receiver's ID (say, Bob), and its own segment. A malicious AD can store this information in his own local database. Upon receiving a `reply` or `data` packet, the malicious AD can compare the stored segments from the e-path in the packet, and learn the missing segments from the sender to the receiver. As a result, all `data` packets carrying the same segments would be linked to the same sender-receiver session. On the other hand, when different segments are used in a new session, the AD cannot tell if Bob is still communicating with the same sender, thus achieving session unlinkability.

6.6.3 General Attack Resilience

DoS resilience. Prior anonymity systems are often vulnerable to computational-based DoS due to expensive asymmetric operations for setting up communication paths or storage-based DoS due to stateful forwarding. As a result, they require additional DoS defense mechanisms, such as introduction points [47] or mailboxes [108], as an extra layer of indirection to actively block unwanted requests. On the other hand, LAP is robust against Denial-of-Service (DoS) attacks in many aspects, thanks to its lightweight path establishment and stateless forwarding mechanism. For example, a receiver can filter incoming traffic by selectively announcing paths and frequently updating paths.

A common challenge for all anonymity systems is when an attacker sends untraceable traffic. To prevent such misuse of anonymous communications, an AD can allocate only a small amount of bandwidth for anonymous traffic. To prevent such attacks, we leave it as future work to study the tradeoffs between anonymity and accountability.

Resilience against traffic analysis. Traffic analysis comprises two parts: observing traffic and correlating traffic. Compared to Tor, LAP makes correlations much easier but observations much harder. For correlations, an attacker controlling two or more distinct entities in the network can easily correlate observed packets to estimate their routes, because LAP packets in the same session look the same at each hop. For observations, a Tor attacker controlling all entry and exit relays has a good chance of de-anonymizing Tor traffic. However, the equivalent attack is almost impossible in LAP because the attacker has to compromise all the first-hop ADs.

6.6.4 Resilience against Known Attacks

DoS-based side-channel attacks. In the category of DoS-based side-channel attacks, the approach proposed by Burch and Cheswick [26] for IP traceback could also be applied to trace back an e-path to its origin. The basic idea is to send a large amount of traffic over a link that the e-path may be using. If the link is indeed part of the e-path, one will observe a slowdown of the session using the e-path. By repeating this process, one could eventually trace back the entire path. The essence of the approach is to induce a DoS attack and to use other packets as a side channel to determine the packet flow. Numerous such side channels have been investigated in the literature [121, 21, 60, 31]. Flow watermarking techniques also fall into this attack category, using slight time-based variations to infer which packets belong to the same session [75, 118]—however, this requires multiple observation points in the network. These attacks are possible even on more heavy-weight schemes such as Tor, and naturally our lightweight approach will not offer protection. These attacks, however, require more significant effort than passive observations of network traffic.

Time-based identity inference attacks. A related attack class is time-based identity inference attacks. Specifically, Kohno et al. propose device fingerprinting based on clock skew inferred from TCP timestamps [90]. Since in LAP, TCP headers are not encrypted by default, this attack would apply. However, the standard countermeasures apply as well: end-to-end IPsec tunnel, perturbation of TCP timestamp, etc. Another potential location leak is round-trip-time (RTT) based location inference, where the observation is that the lowest observed RTT induces an upper bound on the distance of the other party. Consequently, ACK packets, for example, may need to be delayed to increase the anonymity set.

TTL-based attacks. Finally, in the case of LAP used on IP-based networks, we need to defend against a TTL-based attack: by sending a LAP packet with a small TTL, the TTL may expire while a router within the e-path forwards the packet, which in turn would trigger an ICMP message sent to the source address. Fortunately, the first router in the e-path sets the IP source address to its own address, thus the attacker would not receive the ICMP error message.

6.7 Evaluation

In this section, we evaluate the performance of LAP in terms of latency and throughput. Specifically we compare three systems: LAP-disabled (no anonymity), LAP-enabled (intermediate anonymity), and Tor (high anonymity). Our results show that LAP improves anonymity with a negligible overhead (i.e., lightweight) and is more efficient compared to high anonymity systems like Tor.

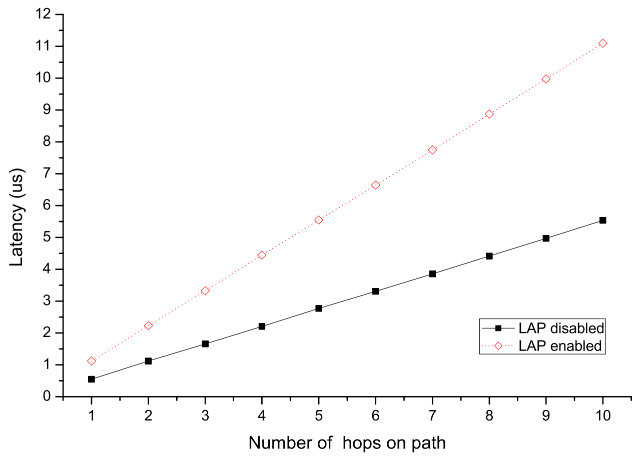


Figure 6.9: Average latency with LAP disabled and LAP enabled.

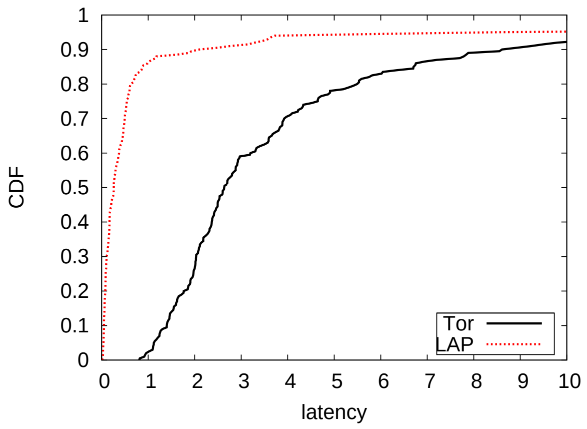


Figure 6.10: Latency comparison of LAP and Tor using the real Internet topology.

LAP implementation. We implement basic routing and forwarding elements based on Click software routers[12] to support packet-carried forwarding state (LAP-disabled). We extend the prototype to further support encryption/decryption of LAP (LAP-enabled). The only overhead that LAP introduces for an e-path construction per AD hop is the extra packet space needed for optimal padding, and the time for

[12]The Click Modular Router Project. http://read.cs.ucla.edu/click/

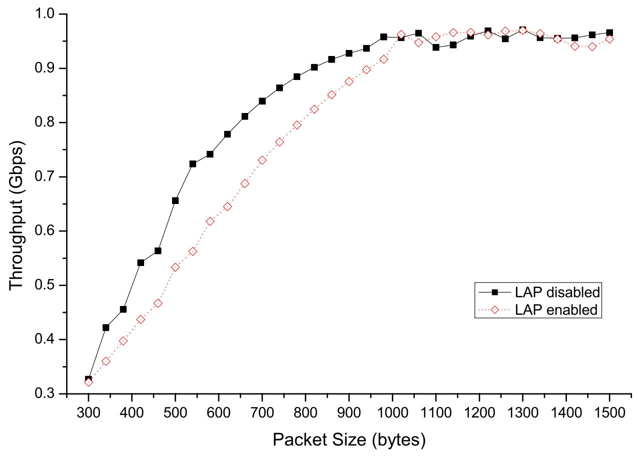


Figure 6.11: Average throughput with LAP disabled and LAP enabled.

a symmetric encryption. This is because packet-carried forwarding state already requires ADs to verify their own routing decisions using MACs. Since routing decisions are carried in each packet, the overhead caused by the forwarding phase for each AD is the time to decrypt its own segment. We show that our software-based implementation of LAP exhibits competitive performance, with an anticipation that LAP will perform even better on dedicated hardware.

6.7.1 Latency Evaluation

We first examine the latency introduced by LAP's cryptographic operations. We then estimate LAP's latency in the real Internet and compare with Tor.

We measure the latency of LAP-disabled and LAP-enabled systems in one LAN network. Each AD is simulated on one machine with 1 Gbps connection to its adjacent ADs. Since our tests are run on a local LAN, the latency is dominated by the cryptographic operations. We implement LAP's encryption/decryption using the AES function in OpenSSL. For the LAP-disabled case, ADs perform forwarding using packet-carried state, which involves one MAC computation using the same AES function. For the LAP-enabled case, ADs verify a MAC and decrypt their own state during forwarding. We run each test 10 times and present the average value. As Figure 6.9 shows, LAP adds a small amount of latency to packet processing. In our software implementation, this is on the order of microseconds, but a hardware implementation would shrink the extra decryption time to nanoseconds.

We also compare the latency experienced by LAP and Tor users using the real Internet topology as follows: we estimate LAP's latency based on the actual Round-Trip-Time of receiving HTTP packet headers and the estimated latency overhead of LAP cryptographic operations. For Tor, we measure latency using the actual Tor network. Specifically, we measure the latency with and without Tor between 10 geographically distributed machines and the top 200 university websites reported by Alexa[13], and also resolve the URLs of these sites in advance to exclude DNS lookup time. We use university sites as they are less likely to redirect traffic based on source addresses (in contrast to popular commercial sites). As Figure 6.10 shows, LAP users experience significantly lower latency compared to Tor users: 90% of LAP requests finish in less than one second, while most ($> 99\%$) of Tor requests take more than one second.

6.7.2 Throughput Evaluation

We evaluate LAP's impact on throughput using Netperf 2.5.0[14] with synthetic traffic of different packet sizes. Figure 6.11 shows the average throughput of LAP-disabled and LAP-enabled systems. We observe that the throughput grows with packet size for both cases. In particular, the throughput for the LAP-enabled case is slightly lower than the one for LAP-disabled, since it takes more time for LAP to process a packet than to simply forward it. However, the difference in these throughput is very small or even negligible, especially when the packet size is beyond 1 KByte. This result confirms that LAP has a small impact on router performance.

We also compare the throughput between LAP and Tor using a small testbed that runs LAP as well as a private Tor network with three Tor relays. For this evaluation, we set four machines in the testbed to be connected among each other using 1-Gbps links, each machine dedicated to be a source, a destination (file server), an intermediate machine running three Tor relays, and a Tor directory server. With this testbed, we measure the average throughput of a client machine that is downloading a 10-GB file from the file server for LAP and Tor. When downloading a 10-GB file using the Tor network, the client's average throughput is $\mu = 50.79$ Mbit/s ($\sigma = 1.41$). With LAP, $\mu = 939.50$ Mbit/s ($\sigma = 32.76$), showing a significant throughput increase.

To summarize, the overhead that LAP imposes is minor, which makes LAP suitable for practical deployment. In particular, at the cost of a small throughput decrease, LAP can improve the anonymity in current IP networks.

6.8 Summary

Current anonymous communication systems achieve a high level of anonymity against a strong attacker model, but pay a dear price in terms of overhead: high communication latency with high in-network computation and storage state. Especially the high latency causes the Internet browsing experience to endure a significant slowdown.

[13] `http://www.alexa.com/topsites/`
[14] `http://www.netperf.org/netperf/`

Anonymous communication would thus be more usable with reduced overhead. Indeed, we believe that many users can live with a relaxed attacker model, as they can trust their local ISPs but want protection from tracking by ISPs that are further away (potentially in other countries with different privacy laws) and from tracking by websites. Given such a weaker attacker model, we attempt to provide source and destination anonymous communication, session unlinkability, and location privacy at a very low overhead, barely more than non-anonymous communication.

In this framework, our approach is simple yet effective: by leveraging encrypted packet-carried forwarding state, ISPs that support our protocol can efficiently forward packets towards the destination, where each encrypted ISP-hop further camouflages the source or destination address or its location.

Although encrypted packet-carried forwarding state is currently not supported in IP, we design simple extensions to IP that could enable this technology. In particular, our approach is even more relevant in future network architectures, where the design can be readily incorporated.

This new point in the design space of anonymity protocols could also be used in concert with other techniques, for example, in conjunction with Tor to prevent one Tor node from learning its successor. Despite weaker security properties than Tor, we suspect that LAP contributes a significant benefit towards providing topological anonymity, as LAP is practical to use for all communication.

Chapter 7

Integration and Discussion

The goal of this dissertation is to design efficient defense mechanisms that provide meaningful and actionable availability guarantees for end-to-end packet delivery despite link-flooding DDoS and address-based selective dropping attacks.

To achieve this goal, we explore three aspects of DDoS defense—bandwidth guarantees, waiting time guarantees, and flow monitoring—in Chapter 3 through Chapter 5. We present STRIDE, a DDoS-limiting architecture that provides bandwidth guarantees, and RainCheck Filter, a primitive that bounds the maximum waiting time to get through a bottleneck on the communication path. We also present LFD, an anomaly detection algorithm that efficiently catches large flows without keeping per-flow state. We then explore mitigation to address-based selective dropping using anonymous communication in Chapter 6 and present LAP, a primitive that hides the network addresses and topological locations.

This chapter describes an integration of the four defense mechanisms on top of SCION (reviewed in Section 2.2). We first summarize the adversary model and assumptions that are required for achieving availability guarantees. We then show that the integrated solution offers waiting time and bandwidth guarantees in the presence of DDoS and address-based selective dropping adversaries. We conclude this chapter with discussions such as whether anonymity and DDoS defense can be achieved simultaneously.

7.1 Integration

We review the adversary model in the context of SCION.

7.1.1 Adversary Model

Both endhosts and ADs can misbehave. We aim to address the following attacks:

Link flooding by malicious endhosts. Malicious endhost machines can collaboratively send excessive traffic to overload a network link, thus attempting to crowd out legitimate flows. This is referred to as link flooding or a DDoS attack against the infrastructure.

Address-based selective dropping by malicious ADs. We consider malicious ADs that drop packets selectively based on the source, destination, or both addresses. For example, some ISPs are known to target packets from or to rival entities.

Selective dropping by content, service types, timing, etc. is outside the scope of this work. Techniques such as end-to-end encryption can prevent selective dropping based on packet contents. Several host-based tools exist to detect selective dropping by service types and timing, yet there is no efficient solution to mitigate or prevent such attacks. We leave them as future work.

Unlike selective dropping, random dropping can be mitigated in several ways. For small-scale random dropping (i.e., when the dropping rate is about the same as the natural loss rate), a common technical defense is to add redundancy such as multipath, retransmission, or error correcting codes. Large-scale random dropping is likely to severely impact a wide area of entities both within and outside the attacker's ISD, and can be easily detectable using monitoring tools such as traceroute. Once the random dropping is detected, victims in the same ISD can penalize the malicious AD, for example by switching to a different provider or reporting the AD for revocation.

Other types of data-plane attacks (e.g., active packet injection, packet corruption, header modification, and forwarding path alteration) are beyond the scope of this work. Research in fault localization [179] and path authentication [126] can help mitigate such attacks, but further investigation is needed to explore their achievable availability guarantees.

Assumptions. We assume that each endhost trusts its provider AD(s) to never leak its customers' information or maliciously drop their packets. If the first-hop is not trusted, the endhost has to either switch to a different provider or adopt advanced solutions such as censorship-resilient tools.

While ADs may be unable to remove all bots due to politics or technical issues, we assume that benign ADs are cleaner than malicious ADs in the sense that benign ADs contain fewer bots and are willing to remove compromised machines once they are detected.

Since we focus on attacks against AD-level data forwarding, we assume that each AD can resolve congestion and resource contention within its own domain. We discuss technical solutions to address intra-AD congestion and resource contention in Section 7.3. We also assume that ISD Cores are interconnected as a clique, and ISD Cores are congestion-free.

Table 7.1: Forwarding state for different path/channel types. $Enc_k(m)$ means encrypting m using key k, and $MAC_k(m)$ is the Message Authentication Code (MAC) of m using k. A PTS derives the MAC key based the current short-term key, and derives the encryption key based on the current short-term key and a nonce. Nonces are used to ensure that encrypting the same message twice yields two different ciphertexts with high probability. A_{i-1}^T is the MAC generated by the previous hop.

Type	Forwarding State	Message	Authentication
BE	$O_i^{BE} = \{M_i^{BE}, A_i^{BE}\}$	$M_i^{BE} = ingress_i \| egress_i$	$A_i^{BE} = MAC_{ka_i}(BE \| M_i^{BE} \| A_{i-1}^{BE})$
EBE	$O_i^{EBE} = \{\chi_i^{EBE}, A_i^{EBE}\}$	$\chi_i^{EBE} = Enc_{ke_i}(M_i^{BE})$	$A_i^{EBE} = MAC_{ka_i}(EBE \| \chi_i^{EBE} \| A_{i-1}^{EBE})$
S	$O_i^S = \{M_i^S, A_i^S\}$	$M_i^S = ingress_i \| egress_i$	$A_i^S = MAC_{ka_i}(S \| bw \| exp\_time \| M_i^S \| A_{i-1}^S)$
ES	$O_i^{ES} = \{\chi_i^{ES}, A_i^{ES}\}$	$\chi_i^{ES} = Enc_{ke_i}(M_i^S)$	$A_i^{ES} = MAC_{ka_i}(ES \| bw \| exp\_time \| \chi_i^{ES} \| A_{i-1}^{ES})$
D	$O_i^D = \{M_i^D, A_i^D\}$	$M_i^D = ingress_i \| egress_i$	$A_i^S = MAC_{ka_i}(D \| bw \| exp\_time \| M_i^S \| A_{i-1}^D)$
ED	$O_i^{ED} = \{\chi_i^{ED}, A_i^{ED}\}$	$\chi_i^{ED} = Enc_{ke_i}(M_i^D)$	$A_i^{ED} = MAC_{ka_i}(ED \| bw \| exp\_time \| \chi_i^{ED} \| A_{i-1}^{ED})$

7.1.2 Building Blocks

Path Translation Service (PTS). Each AD provides a new service called Path Translation Service (PTS) to convert between different types of forwarding state. Different paths/channels provide different properties and guarantees (which are summarized in Section 7.2).

In this context, a path indicates an authenticated cryptographic representation (i.e., a MAC) of AD-level forwarding information, while a channel indicates that of host-to-host forwarding information. Path translation represents the process of recreating the cryptographic representation for adding, updating, or hiding some information. Note that each PTS handles its corresponding part of a path, and a full translation requires the collaboration of all PTSes on the path.

Recall that best-effort (BE) and static (S) paths are half-paths. Up-paths and down-paths are half-paths with an emphasis on the directionality. A dynamic channel (D) is an end-to-end channel, and the dynamic channel setup request has to traverse an end-to-end path that is a combination of an up-path and a down-path. Paths/channels can be encrypted, resulting in encrypted BE paths (EBE), encrypted static paths (ES), and encrypted dynamic channels (ED). Consider a PTS for AD_i, and O_i^T is AD_i's forwarding information for path/channel type T, where $T \in \{BE, EBE, S, ES, D, ED\}$. Table 7.1 summarizes the forwarding states for different path/channel types.

The PTS in AD_i converts between different types of forwarding states and supports the following conversions:

1. Unencrypted BE to encrypted BE: $O_i^{BE} \Rightarrow O_i^{EBE}$

2. Unencrypted BE to unencrypted static: $O_i^{BE} \Rightarrow O_i^S$

3. Unencrypted BE to encrypted static: $O_i^{BE} \Rightarrow O_i^{ES}$

4. Unencrypted BE or static to unencrypted dynamic: $O_i^x \Rightarrow O_i^D$, where $x \in \{BE, S\}$

5. Encrypted BE or static to encrypted dynamic: $O_i^x \Rightarrow O_i^{ED}$, where $x \in \{EBE, ES\}$

LAP to prevent address-based selective dropping. If an endhost is discriminated against by an AD in the same ISD, the endhost can report this malicious AD for revocation or punishment once it detects such an attack. However, if the endhost is discriminated by an AD in a different ISD, the endhost may be unable to trigger revocation of the malicious AD even when the misbehavior is detected because the AD is under a different jurisdiction. The endhost instead has to adopt a prevention- or resilience-based defense rather than a detection-and-then-recovery defense.

We observe that anonymization downgrades an address-based selective dropping attack to a random dropping attack, which can be easily mitigated. Specifically, encrypted up-paths and down-paths prevent an intermediate AD from selectively dropping packets based on the source and destination addresses, respectively.

Address-based selective dropping attacks can be subtle and difficult to detect, such as when a bottleneck AD selectively drops in the case of congestion (since dropping during congestion is a legitimate behavior). Fortunately, using encrypted paths also prevents the bottleneck AD from dropping selectively, thereby addressing this challenging attack.

For scalability, the sender usually uses one encrypted path for sending multiple packets (e.g., packets associated with the same session). This information allows a malicious AD to link packets in the same session and selectively drop a certain session without knowing the source or destination. To balance security and scalability, a security-sensitive sender can (1) request a new encrypted path more frequently or (2) obtain multiple encrypted paths and use them in turn. We discuss in detail how to improve the level of unlinkability while preserving scalability and DDoS resilience in Section 7.3.

RainCheck Filter to bound the waiting time of requests. Because the ISD Cores are assumed to have no internal congestion, the combination of a static up-path and a static down-path results in at most one bottleneck (i.e., at the top of the down-path) on the AD-level path. The bottleneck AD enables a RainCheck Filter when it experiences congestion. To limit the impact of remote bots, we consider a *path-based fairness model* based on the static up-paths. The raincheck message format is updated to include necessary information for path-based rate limiting:

$$\rho_{path,rid} = m \| MAC_k(m \| path \| rid), \text{ where } m = n \| ts \| t_{start} \| t_{end}. \tag{7.1}$$

The message authentication code is computed using the bottleneck AD's secret key k. Also, *path*, *rid*, n are included to ensure there is no more than n requests in virtual queue for this path. The request id (*rid*) is given by the sender for duplicate detection. Once a request from a path i is accepted, at most $n_i - 1$ additional rainchecks of path i can be renewed or accepted during Δ.

In this scenario, $\frac{N}{\sum_i B_i} \leq \Delta \leq \frac{M}{R_s}$, where M is the available memory for RainCheck Filter at the bottleneck AD, R_s is the request forwarding rate, $N = \sum_i n_i$ is the maximum number of requests in the virtual queue, and B_i is the bandwidth of static up-path i. By setting $B_i \geq \frac{n_i}{\Delta}$, we ensure that rainchecks can be renewed in time. The AD can set n_i to be proportional to B_i, and the ratio is a system parameter determined based on how much bandwidth RainCheck Filter is allowed to consume.

Consider an example with two static up-paths, Path A and Path B, that can send n_a requests and n_b requests per time, respectively. To bound the maximum waiting time by $\frac{n_a+n_b}{r}$, where r is the rate of the congested static down-path, the bottleneck AD ensures that if a request from A is accepted at time t, then it accepts or renews at most $n_a - 1$ more requests during the sliding time window $[t, t + \Delta)$. A similar check is applied to Path B. Note that the waiting time depends on the total capacity of static up-paths, but is independent of the number of bots in other source ADs.

Let $n(A, req)$ be the number of requests that (1) have a higher priority than request *req* and (2) are from Path A. The purpose of this check is to make sure that $n(A, req) \leq n_a$ at any time and is non-increasing over time. This check can be implemented without keeping per-request state. For example, the AD can keep a table of recently accepted requests (indexed by a path ID) where each entry in the table consists of a path ID, expiration time, and a counter. Outdated entries (i.e., entries whose expiration times have passed) are removed. The AD refuses to accept or renew a request from Path A if the corresponding counter exceeds n_a. Otherwise, when the AD decides to accept the request at time t, it increases the corresponding counter value (if the entry already exists) or adds a new entry (Path A, $t + \Delta$, 1).

RainCheck Filter can also be used at bottleneck ADs on the BE down-paths. The same check is applied to bound the waiting time under the path-based fairness model. Following the above example, we can bound the maximum waiting time by $\sum_{AD_{bn} \in bottlenecks} \frac{n_a+n_b}{r_{bn}}$ on a BE down-path when the request extends a partial dynamic channel to overcome the bottlenecks one by one, as described in Section 3.6.2.

Returned rainchecks have to be protected from DDoS attacks as well. The sender can use either a private static path or a partially established dynamic channel for getting the rainchecks. The rainchecks can also use a public static path that prioritizes rainchecks over the other packets.

Monitoring granularity. We consider four levels of monitoring, from coarse- to fine-grained. With exception of flow-level monitoring, these levels monitor flow aggregates grouped by ISDs, ADs, or paths. Flows in the same aggregate share the same fate, such as being punished together. For example, in ISD-level monitoring, the whole ISD is a fate-sharing group and will be punished as a whole if any of its ADs misbehave. To avoid collateral damage due to fate sharing, ISDs and ADs are motivated to better manage their domains and adopt finer-grained monitoring techniques.

1. **ISD level.** Each ISD monitors the traffic aggregate to/from another ISD. Two ISDs can negotiate the bandwidth limits specified in their inter-ISD contractual agreement. For example, ISD A can agree to allocate 5% of capacity on each static down-path for ISD B's traffic. Such limits are especially important when one ISD is unable to verify the claim (e.g., the priority of a packet) made by entities in another ISD.

2. **Neighboring-AD level.** Each inter-AD link has a predefined bandwidth division based on the contractual agreement between these two ADs, e.g., 30% static, 50% dynamic, and the unused part goes to best effort. Each AD monitors the static and dynamic traffic aggregates to ensure their compliance with the respective limits on the link. In addition, ADs can refine the monitoring accuracy by checking against the reserved amounts (e.g., 70 out of 100 Mbps dynamic bandwidth is currently reserved as dynamic paths) rather than the predefined limits.

3. **Path level.** Path-level monitoring identifies static paths consuming more bandwidth than their reserved amounts. We assume that each path is associated with a unique path ID, which can be embedded in the header or derived from the forwarding information. The ISD Cores and the ADs at the joint of two half-paths perform per-path monitoring to enforce the bandwidth allocation for each static path.

4. **Flow level.** Flow-level monitoring aims to catch dynamic flows violating their allocations. Similarly, we assume that each flow is associated with a unique flow ID, which can be embedded in the packet header or derived from the forwarding information. While keeping per-flow state for monitoring is feasible at the edge of the network (i.e., first-hop ADs), it incurs high overhead at intermediate ADs. Per-flow monitoring by the first-hop ADs only may still be insufficient, since some ADs may not trust the first-hop ADs for correct monitoring (e.g., an AD may not trust the first-hop ADs in a different ISD). Probabilistic flow monitoring is more scalable than per-flow monitoring, yet suffers from inaccuracy and detection delay: (1) probabilistic monitoring implies probabilistic bandwidth guarantees, which are weaker than deterministic bandwidth guarantees, and (2) probabilistic monitoring often results in delayed detection, causing collateral damage to benign flows. In Chapter 5, we propose a new setting that trades the level of exactness for scalability, thus enabling the immediate detection of every large flow and perfect protection of every small flow. We discuss how this new setting helps reduce collateral damage and preserve end-to-end availability guarantees below.

LFD to enforce flow bandwidth allocation. We explain how LFD, a deterministic flow monitoring algorithm supporting immediate and almost-exact large flow detection, can be applied to perform real-time enforcement without keeping per-flow state. Recall that LFD considers a relaxed exactness model: all flows violating a high-bandwidth threshold (defined by a leaky bucket) TH_h will be detected immediately, while every flow complying with a low-bandwidth threshold TH_l is protected and never falsely caught. Medium flows (those flows which are in between the two configurable thresholds) can be identified probabilistically using existing techniques such that there is a reasonable probability of catching medium flows. Consequently, a coward attacker who hates taking any risk is trapped and forced to stay within TH_l. An aggressive attacker may take some risk to go beyond TH_l but is still trapped and forced to stay within TH_h. This distinct property allows an AD to estimate the minimum capacity for achieving desired bandwidth guarantees due to DDoS attacks.

Particularly, to avoid undesired dropping despite DDoS flows violating their bandwidth limits, an AD over-provisions based on a ratio R_{op}, where $R_{op} \triangleq \frac{TH_h}{TH_h}$. To see why this works, consider an AD with $D \geq n \cdot TH_l$ units of dynamic bandwidth for supporting n flows. Since flows violating TH_h could be caught immediately, the total bandwidth consumption of these n flows is bounded by $n \cdot TH_h$. In other words, an AD with an over-provisioned capacity $\geq R_{op} \cdot D \geq n \cdot TH_h$ can robustly prevent collateral damage, provided such immediate and almost-exact flow-level monitoring is used.

7.1.3 Integration on top of SCION Architecture

For ease of presentation, we describe each protocol step in this integrated solution.

Half-Path Setup

① **Path discovery and bandwidth announcement:** The bandwidth announcement part is the same as in STRIDE. In short, Path Construction Beacons (PCBs) are periodically initiated by the ISD Core and are propagated from the ISD Core to endpoint ADs. Each intermediate AD adds information to the PCBs regarding the available bandwidth.

To enable LAP for selective dropping defense, the Path Server in each AD also stores one BE path for each route that the AD has propagated to the downstream ADs.

② **Path translation:** Upon receiving a BE half-path, the endpoint AD can decide whether to translate the BE path to an encrypted BE path, an unencrypted static path, or an encrypted static path, each of which supports different properties. If a path translation request is issued, the PTSes on the BE path will translate the path. For encrypted static paths, the admission control can only be done based on the ingress point and the previous-hop AD.

③ **Confirmation:** Once successfully translated, ADs send the confirmation using the new path. The endpoint AD can register the new path at the Path Server for public usage.

Static and BE Channel Setup

④ **End-to-end path selection:** We consider two types of end-to-end channels from the sender to the receiver: (1) combined channels that are built from half-paths, and (2) dynamic channels. Combined channels are low-capacity or best-effort channels that are mainly used for sending dynamic channel setup requests. Dynamic channels are high-capacity channels.

Based on the sender's requirement, the source AD constructs a combined channel consisting of (1) the sender's endhost identifier (EID), (2) an AD-level up-path, (3) an AD-level down-path, and (4) the receiver's EID. While in principle the source AD selects the AD-level path on behalf of the sender, we also allow the sender to explicitly specify the AD-level path (e.g., a receiver provides a private down-path exclusively for this sender).

To support sender anonymity, the source AD encrypts the sender's endhost identifier (EID) while forwarding the packets. To support receiver anonymity, the receiver first requests a pseudonym from the destination AD, where the pseudonym can be an encrypted EID created by the destination AD. The receiver wishing to run a hidden service publishes an encrypted down-path with its own pseudonym at the Path Server in the ISD Core.

Specifically, the sender includes in the request packet the following information: (1) the sender's endhost identifier (EID), (2) the receiver's EID, or pseudonym plus encrypted down-path, (3) indicator of sender anonymization, (4) desired type of waiting time guarantees, and (5) desired amount of dynamic bandwidth where zero indicates the maximum possible.

Seeing this host-generated request packet, the source AD replaces it with a properly formatted dynamic channel setup request packet with the following fields: (1) the sender's EID or pseudonym, (2) the receiver's EID or pseudonym, (3) an AD-level path (which satisfies the specified anonymity and waiting time requirements), (4) the desired amount of dynamic bandwidth, and (5) the enablement, if necessary, of RainCheck Filter (it is enabled if the sender requires a maximum waiting time guarantee).

Dynamic Channel Setup

⑤ **Dynamic channel setup request:** Depending on the types of combined channels and the strategy used at the bottleneck to resolve congestion, the sender achieves various waiting time guarantees for establishing a dynamic channel, as summarized in Table 7.2.

Table 7.2: Waiting time guarantees. The destination ISD allocates a certain portion of bandwidth for each source ISD, as described in Section 3.6.1. For random dropping, we assume the probability of getting through the bottleneck is inversely proportional to the total incoming static traffic. For BE down-paths, we assume that partial dynamic channels (viz., Section 3.6.2) are in use and that the request can be delivered before the partial dynamic channel expires.

Up-path	Down-path	Congestion handling	Guarantees
Static	Static (private)	N/A	MWT guarantees (no wait)
Static	Static	RainCheck Filter	MWT guarantees linear to total BW of static up-paths in the source ISD
Static	Static	Random dropping	Probabilistic waiting time linear to total BW of static up-paths in the source ISD
Static	BE	RainCheck Filter	MWT guarantees linear to total BW of static up-paths in the source ISD times # of bottlenecks
Static	BE	Random dropping	Probabilistic waiting time linear to total BW of static up-paths in the source ISD times # of bottlenecks

⑥ **Dynamic-class bandwidth allocation:** As in STRIDE, ADs allocates the dynamic bandwidth so that every request traversing a static path is guaranteed a lower bound on the reservable dynamic bandwidth. Section 3.7.2 provides an example of a simple dynamic allocation policy. Note that when paths are encrypted, an AD can only perform admission control based on the previous and next hops.

An AD encrypts its part of the dynamic channel if the counterpart of the combined channel is encrypted.

⑦ **Guaranteed data transmission:** The dynamic channel isolates a flow from the others to protect it from being crowded out by attack traffic during DDoS attacks. In other words, the network will protect a flow so long as the flow stays within the specified limit. Such flow-level isolation relies heavily on the real-time detection of flows violating their allocations during link congestion. We use LFD for this purpose, as explained in Section 7.1.2.

7.2 Availability Guarantees

We analyze the bandwidth and waiting time guarantees using an incremental approach: we add one mechanism per time to SCION, and state the guarantees and relevant assumptions at each stage. The guarantees are strong as they are independent of attackers outside the source and destination ADs. Table 7.3 summarizes the results.

Chapter 3 presents the details of STRIDE's guarantees. Note that to achieve these guarantees, STRIDE assumes that (1) ADs do not discriminate against traffic, and (2) ADs can monitor and regulate each flow.

Adding RainCheck Filter strengthens the waiting time guarantees, as the bounds change from probabilistic to deterministic.

Table 7.3: Guarantees and assumptions. We assume that the source and destination ADs are clean, and discuss how to handle intra-AD DDoS attacks in Section 7.3.

	STRIDE	STRIDE + Raincheck	STRIDE + Raincheck + LFD	STRIDE + Raincheck + LFD + LAP
Waiting time				
Static/Private static	MWT guarantees (no wait)			
Static/Public Static	expected $\propto \frac{\sum \text{ static up-path BW in srcISD}}{\text{down-path BW}}$	MWT $\propto \frac{\sum \text{ static up-path BW in srcISD}}{\text{down-path BW}}$		
Static/BE	expected $\propto \sum_{bn} \frac{\sum \text{ static up-path BW in srcISD}}{\text{bottleneck (bn) BE BW}}$	MWT $\propto \sum_{bn} \frac{\sum \text{ static up-path BW in srcISD}}{\text{bottleneck (bn) BE BW}}$		
Others	No guarantees			
Reservable bandwidth				
Static/Static	Reservable BW $\propto \min_{l \in path} \frac{l\text{'s dynamic BW}}{l\text{'s static BW}}$			
Static/BE	Reservable BW $\propto \min_{l \in path} \frac{l\text{'s dynamic BW}}{l\text{'s BE BW}}$			
Others	No guarantees			
Assumptions	Benign ADs, per-flow monitoring		Benign ADs	N/A

Adding LFD allows us to efficiently identify flows violating the allocated bandwidth, thereby reducing monitoring overhead. LFD removes the need to perform expensive per-flow monitoring at intermediate ADs.

Adding LAP prevents selective treatment based on network identifiers. When LAP is enabled, ADs can only identify misbehaving flows that violate the allocations, but not the actual sender(s). Interestingly, though the actual sender of a misbehaving flow is unknown, ADs can still notify the source AD (which is also unknown) of such misbehavior by sending a warning message back to the source AD along the reversed encrypted channel.

7.3 Discussion

Tradeoffs between anonymity and DDoS defense. Anonymization, or indistinguishability in the sense that the attacker cannot tell which flow is the real target, can prevent address-based selective dropping, thereby improving Internet availability in one aspect via indistinguishability between flows. This dissertation aims to achieve topological anonymity, which is used to defend against address-based selective dropping in the presence of a constrained adversary who can leverage topological information only and cannot compromise the first-hop AD of a victim. In other words, a stronger attacker who exploits additional information (e.g., timing correlation) or compromises the first-hop AD may be able to de-anonymize and therefore selectively block the victim. Also, topological anonymity is insufficient to prevent selective dropping based on other criteria, such as traffic volumes, timing, and packets linked the same anonymized flow.

Generally speaking, anonymity and DDoS defense seem to in contradiction because the attacker can evade DDoS detection through anonymization. However, given the weak notion of anonymity (i.e., topological anonymity) considered in this dissertation, it is in fact possible to achieve topological anonymity and DDoS defense at the same time.

To examine whether they are mutually achievable in our design, we first review the high-level operations of our DDoS defense mechanisms, with an emphasis on the required identification information. To mitigate DDoS attacks, we divide bandwidth into isolated channels and check whether packets using the same channel consume more than the allocation. This requires the ability to distinguish legitimate flows (i.e., flows complying with their allocations) from malicious flows (i.e., flows violating their allocations), so that we can cut off malicious flows without hurting legitimate flows. In other words, our DDoS defense mechanisms require knowing the flow identifiers and their corresponding allocations, but do not require knowing the topological location of the sender or receiver. In particular, bandwidth allocation depends on the previous and/or the next hop AD. The bandwidth enforcement is behavior-, rather than address-based. Putting together, we observe that because topological anonymity only hides the addresses in packet headers, packets belonging to the same flow can still be linked together based on their anonymized flow identifier, which enables flow-level monitoring based on flow behaviors (i.e., whether a flow violates its bandwidth limit). In addition, our anonymous forwarding mechanism allows an intermediate AD to inform the source of a violation without knowing who the sender is.

In a realistic setting where the adversary attempts to de-anonymize the victim by correlating various sources of information, the level of topological anonymity may be reduced over time as the adversary gathers more information. One typical solution to this problem is to change the anonymized identifier from time to time such that the adversary does not have enough time to gather sufficient information for de-anonymization. To be effective, the new identifier has to be unlinkable from the old one. While the level of anonymity increases with the frequency of the identifier change, quickly changing the identifier causes high overhead: static paths have to be updated frequently, which means the signaling mechanism has to be fast. The tradeoff should be made based on the adversary power and the desired performance.

We discuss possible approaches and limitations for achieving two common forms of unlinkability.

- **Between-flow unlinkability:** No adversary can determine whether two flows originate from or are destined to the same topological region with non-negligible probability. One method to achieve between-flow unlinkability is to send each request using a different encrypted combined channel, which requires the source and the destination ADs to have a large number of encrypted half-paths. However, maintaining too many static half-paths increases the complexity of traffic monitoring, and each static half-path would have an extremely low capacity, which makes it impractical to use.

Thus, for static half-paths, senders in an AD have to reuse some static half-paths for establishing flows, trading the level of between-flow unlinkability for scalability. Alternatively, the source and destination ADs can choose to use encrypted BE half-paths, which are not subject to the same complexity and practicability concerns; however, the high level of between-flow unlinkability is obtained at the cost of the waiting time and reservable bandwidth guarantees.

- **Within-flow unlinkability:** No adversary can determine whether two packets belong to the same flow with non-negligible probability. One way to achieve within-flow unlinkability would be to have each packet to take a different encrypted dynamic channel, and none of which can be linked to each other. Changing the channel for each packet is inefficient and unnecessary, because if there is a secure method to send a dynamic channel setup request, then the sender can simply use the method to send the packet. Consequently, providing within-flow unlinkability at a per-packet level may be an overly strong property. Instead, senders may benefit from a weaker property where some, but not all packets, are linkable. For example, the sender can divide packets in a flow among multiple dynamic channels. This approach is similar in spirit to the concept of switching between a set of anonymous certificates for long-term unlinkability in ad hoc networks [150].

We leave it as future work to explore whether anonymity and DDoS defense can coexist under different anonymity notions and types of DDoS defense.

Communication via shortcuts. In addition to using paths that go through the ISD Cores, a source in SCION can also opportunistically use shortcuts (i.e., end-to-end paths that bypass ISD Cores) when they exist. A shortcut is represented by an up-path, a down-path, and a crossover point that indicates where the packets should switch from the up-path to the down-path. The crossover point can be either a common AD on both of these two half-paths or a peering link connecting them.

While this dissertation focuses on designing mechanisms to achieve waiting time and bandwidth guarantees for communication going through the ISD Core(s), we would also like to discuss what availability guarantees can be achieved when the source chooses to use shortcuts.

First, we note that the source can still establish bandwidth-guaranteed dynamic channels once the initial request gets through the bottlenecks and reaches the destination, which means the bandwidth guarantees remain the same regardless of the use of shortcuts. However, waiting time guarantees are affected for two reasons:

- Congestion at the crossover points introduces additional bottlenecks. Unlike the ISD Cores, which are assumed to be congestion-free by design, traffic may encounter congestion within the crossover

AD or on the crossover peering link. Congestion at the crossover point introduces additional bottlenecks on the end-to-end path and thus weakens the waiting time guarantees.

- Enabling shortcuts may result in multiple bottlenecks on the down-paths. Imagine a scenario in which multiple tributaries flow into a main stem. The main stem may overflow at any of the confluences if it does not have sufficient capacity. In our case, the static down-path from the ISD Core to the destination is the main stem, and the shortcuts are the tributaries. In the current design, if static traffic on the shortcuts is allowed to merge into the static down-path, we lose the no-bottleneck property on the static down-paths, and thus the waiting time guarantees for communication via ISD Cores are again weaken.

To mitigate these problems while supporting shortcuts, we propose the following modifications. First, when activating a static half-path, the endpoint AD has to specify which crossover point(s) it would like to use. Second, for each AD x_i (ordered from the edge to the ISD Core), the activated static bandwidth between x_i and the core decreases when x_i is or connected to a crossover point. In other words, the half-path increases its downstream capacity for each crossover point. Third, since there is no static allocation within the crossover AD or along the crossover peering link, the source gets a fair share based on the static up-paths. Fourth, when merging crossover traffic into the down-path, the AD ensures that the crossover traffic uses no more than the extra capacity, such that the traffic flowing in the main stem does not get affected. This construction ensures (1) fair sharing at a congested crossover point and (2) at most one bottleneck on the static down-path.

Alternatively, ADs could try to avoid static-class congestion at the crossover point. This is possible if a peering link is well-provisioned such that its static-class capacity is at least as much as the sum of all the static up-paths of the previous AD. Similarly, a crossover AD can avoid internal congestion for static-class traffic if its internal capacity is high enough to directly forward packets from every up-path to some down-path. However, these assumptions require provisioning for the worst-case scenario and therefore may result in underutilization for most of the time, which is undesirable for ISPs.

Congestion within ISD Cores. To achieve waiting time guarantees, this work assumes that the ISD Cores are never overloaded by static-class traffic. To justify this assumption, we describe one possible approach here.

Since ADs in ISD Cores are bound by contractual agreements, we could have a policy that every newly joined AD has to establish direct links to each of the current members in the ISD Cores. This requirement is feasible because the ISD Core topology, which contains the tier-1 ISPs, is likely to be small. There appears to be fewer than 15 tier-1 ISPs in the current Internet. Moreover, each of these direct links should be able to support at least a certain level of static-class bandwidth, say, C. Such requirements on link capacity are commonly seen in current peering agreements as well. When advertising available bandwidth, ADs in the ISD Core ensures that the total available static-class bandwidth does not exceed C. ISD Cores should require direct links to each other. For each pair of ISD Core, an upper bound is set on the static-class traffic between the two ISD Cores.

What are the architectural primitives that help our design? In retrospect, our design demonstrates that SCION's top-down route discovery, isolation domains, and path control are helpful for creating a highly available network. We argue that our design can be applied to any architecture that provides exactly the same primitives as these three. It is a challenging problem to determine whether these primitives are still sufficient when we generalize them and consider generic, rather than SCION-specific, definitions. For instance, it is unclear whether our design would still work on an architecture that provides path control to the source AD only (in contrast to SCION, where both the source and destination ADs can control it). We leave it as future work to formulate these primitives and prove whether these architectural primitives are necessary or sufficient conditions for high availability.

Top-down route discovery enables scalable beaconing. Our design leverages scalable beaconing such that provider ADs periodically advertise the available bandwidth to their customer ADs. Endpoint ADs can then allocate bandwidth along the tree topology. In addition, the tree topology simplifies our bandwidth allocation. Specifically, while end-to-end allocation is hard, allocation is simple along a tree that is rooted at the ISD Core, as the available bandwidth can be split from the root down to each leaf. Without tree-based allocation, it is difficult to allocate bandwidth in the current Internet due to its complexities, which include sending out requests for every flow and splitting bandwidth into many small pieces without any meaningful lower bound.

ADs in the same isolation domain share a common root of trust, which enables explicit trust and accountability for routing within an ISD. Because of this property, misbehaving entities within an ISD can be efficiently detected and then revoked. For example, a malicious AD on the up-path could manipulate the packet annotations such that the down-path is tricked to believe that every packet came from static paths. Within an ISD, such manipulation is easily detectable by the next-hop AD or the ISD Core. On the other hand, an ISD may be unable to detect such manipulation when traffic originates from another ISD because the ISD Core in another ISD may also be malicious. Hence, in our design, each ISD has to set an explicit limit on how much traffic another ISD can send to it, and therefore avoids being fooled by others. However, since each ISD is treated as a single entity to external ADs, inter-ISD guarantees are not as strong as intra-ISD guarantees. In sum, isolation domains allow us to achieve stronger guarantees for intra-ISD traffic than inter-ISD traffic.

In SCION, both source and destination ADs can control which path to use for the end-to-end communication. This path control property allows us to provide diverse guarantees because the endpoint ADs can choose to activate and use different types of paths. We support three bandwidth classes (i.e., static, dynamic, and best effort) and four anonymity types (i.e., no anonymity, sender anonymity, receiver anonymity, and sender and receiver anonymity). In addition to path control, packet carried forwarding state is also useful as it encodes forwarding information in the packet header, such that routers do not need to keep state.

Bandwidth overbooking. Since only a subset of advertised half-paths will be activated, our design allows an AD to advertise more than its actual capacity for improved utilization and relies on explicit activation to prevent malicious ADs from selling more than its capacity. Explicit activation ensures that a static half-path is allocated only if every AD on the path agrees. More concretely, consider an example where AD A advertises 1 unit of static bandwidth to AD B, which in terms advertise 1 unit to both AD C and AD D. This step is legitimate because AD B might observe that only 50% of the bandwidth will be activated based on the past history, but AD B has to reject one of the activation requests when both AD C and AD D want to activate their 1-unit paths. AD B may be greedy and want to accept both of the requests, but such misbehavior is detectable by either its provider or its customer: since AD A only advertised 1 unit, suspicion will be raised when AD B tries to activate 2 units. Moreover, even if AD B manages to collude with AD A, its customers will be aware when it fails to meet the promises.

Practical waiting time bounds. One of our goals is to limit the waiting time, which may be unbounded in the current Internet. Although our design bounds the waiting time through isolation and fair access, the bound may be too high to be practical. To enhance our guarantees, we can adopt techniques that identify attackers or localize faults, as they can block, remove, or avoid the identified attackers or faults. In Section 4.4 we discuss techniques that can reduce the bound and thus strengthen our waiting time guarantees.

Congestion within ADs. This work on improving Internet availability is mainly within the context of inter-AD packet forwarding, and assumes that packets can be reliably forwarded within an AD. We now discuss how to address congestion within an AD.

- **Congestion by intra-AD DDoS attacks.** In an intra-AD DDoS attack, the attack traffic never leaves the AD. Customers expect their provider AD to address intra-AD DDoS attacks, as the attack sources are in the same AD and thus can be efficiently blocked or removed by the AD. As a result, ADs that fail to address internal congestion and that ignore customer needs will likely be kicked out of the market.

- **Congestion in source ADs.** Resource contention among hosts, such as when multiple customers compete for the outgoing link bandwidth, can cause congestion inside the source AD. As in today's Internet, an AD is responsible for resolving resource contention among its customers. An AD can alleviate contention via various technical and policy solutions such as traffic engineering and bandwidth caps.

- **Congestion in destination ADs.** While infrastructure DDoS attacks are on the rise, DDoS attacks targeting servers (also known as server flooding), remain a great threat against Internet-based services, as access links are much easier to overload than backbone links. Specifically, server flooding exhausts the uplink, downlink, or the internal resources of the server.

  The mechanisms proposed in this thesis can address server flooding from two complementary perspectives. First, because of AD-level bandwidth allocation and ISD isolation (provided by STRIDE), attack flows originating from the same AD or ISD have to first compete with each other before they are able to reach the victim. In this architecture, benign flows in a clean AD or ISD have a greater chance of reaching the victim server than they do in the current Internet. Second, as described in Chapter 4, placing RainCheck Filter in front of a bottleneck can bound the waiting time to pass the bottleneck. In the case of server flooding, either the server can run a RainCheck Filter by itself, or the destination AD can install a RainCheck Filter for the victim service at the ingress routers, thus minimizing collateral damage inside the AD.

- **Congestion within intermediate ADs.** By establishing a path or a channel of capacity c from an ingress A to an egress B, the AD agrees to forward packets from A to B along the path or the channel at rate c even during congestion. We note that an AD can accurately assess, provision, and engineer the internal routes because it has full control over the advertised available bandwidth. Each AD can also have its own solutions to meet this requirement and thus avoid internal congestion,

Collateral damage. A DDoS defense system causes collateral damage if implementing this defense negatively affects the level of availability for legitimate flows. Minimizing collateral damage requires the ability to distinguish good traffic from bad traffic based on the behaviors, origins, traffic histories, etc. However, DDoS traffic can mimic legitimate traffic, which makes it extremely difficult to identify attack traffic without false positives, false negatives, or delayed detection, all of which result in collateral damage to legitimate traffic.

One promising approach is to define “legitimate flows” in such a way that legitimate flows will never hurt the system. With this approach, malicious flows mimicking legitimate flows are trapped in a dilemma as they will either be caught or have to behave like legitimate flows that cause no harm at all. For example, in CoDef [102], legitimate flows are defined as flows that respond to requests asking them to slow down or reroute. CoDef’s reroute mechanism enables the target router to resolve flooding without dropping packets when paths are sufficiently diverse. When there is a lack of path diversity, its rate control mechanism ensures that each source AD gets a fair share at the bottleneck link, and thus limits collateral damage to source ADs. In our work, legitimate flows are those that comply with the bandwidth allocation.

Chapter 8

Related Work

To ensure packet delivery and improve Internet availability despite active adversaries, this dissertation explores security mechanisms that mitigate two data-plane attacks: Distributed Denial of Service (DDoS) and address-based selective dropping. This chapter reviews related work in the following relevant research areas:

- Section 8.1: DDoS countermeasures
- Section 8.2: Countermeasures to selective dropping
- Section 8.3: Network traffic monitoring and measurement
- Section 8.4: Topology and routing for high availability

8.1 DDoS Countermeasures

A typical Denial-of-Service (DoS) attack makes a service unavailable to legitimate users by exhausting limited resources[1], which can be the bandwidth of bottleneck links, the CPU time of servers, etc. A Distributed Denial-of-Service (DDoS) attack is a type of DoS attack involving more than one attack source.

Depending on the type of messages used for flooding, DDoS attacks can be referred to as ICMP floods, UDP floods, TCP SYN floods, HTTP floods, and so on. While most DDoS attacks to date aim to paralyze servers, emerging DDoS attacks such as Coremelt [149] and Crossfire [80] attempt to overwhelm an intermediate link on the communication path, thus preventing legitimate users from accessing victim servers through the flooded link.

[1]A DoS attacker can also exploit software vulnerabilities to crash a system remotely. While vulnerabilities can be fixed via software updates, exhaustion-based DoS attacks are difficult to mitigate even when all bugs are removed. Hence, we focus on resource exhaustion attacks in this dissertation.

Traditional DDoS mitigation solutions, such as over-provisioning and anomaly detection (e.g., Intrusion Prevention Systems), are expensive or ineffective, as DDoS attacks grow rapidly in volume and sophistication.

This section reviews the current frontier of research in defending against large-scale and sophisticated DDoS attacks. DDoS defense mechanisms can be largely classified into three categories: (1) filtering, (2) resource allocation, and (3) fair access, which will each be reviewed in turn. One can generally combine approaches in these categories for stronger properties.

8.1.1 Filtering

Filtering approaches [14, 110] install filters against attack sources near their origins (i.e., source ADs) to prevent collateral damage of attack traffic. This would essentially require trust establishment between ADs and would rely on source ADs' cooperation, which would incur substantial overhead for managing flow state and packet inspection. In contrast, STRIDE facilitates natural trust relationships between ADs within the same trust domain. Network capabilities [168, 171] enable destination-controllable flow prioritization and stateless filtering at intermediate routers. To construct a capability, a sender sends a capability request to the destination, and every router on the forward path adds to the packet header a cryptographic authenticator that can only be computed and verified by that router. If the destination grants the request, it returns all the authenticators (representing the capability) to the sender. In general, packets with capabilities take precedence over packets without. However, network capabilities are vulnerable to Denial-of-Capability (DoS) attacks [15]. Portcullis [129] leverages computational puzzles to address the DoC attack but renders a high computational overhead even on benign endhosts. StopIt [110] proposes filtering attack traffic at locations near the attack origin. However, since StopIt requires the receiver to inform the source AD of undesired flows, it cannot mitigate the Coremelt attack where all flows are "desired" by bots sending traffic among themselves. In addition, Related work often focuses on router-based approaches for some sort of fairness at bottleneck links rather than on an architectural approach, and it is unclear how to translate per-link guarantees into end-to-end guarantees.

8.1.2 Resource Allocation

Existing approaches [23, 20] that aim to provide bandwidth guarantees to flows fail in situations where all available bandwidth is exhausted. FLoc [100] differentiates legitimate flows from attack flows to provide differential bandwidth guarantees. Low-rate attack flows, however, often cannot be precisely distinguished from legitimate flows, and therefore the lower bound of bandwidth may not be observed.

Queuing systems are heavily researched in a number of disciplines, particularly computer science and operations research. They are also employed by many DoS defense systems both for efficiently scheduling clients' requests and for identifying/penalizing multiple requests from a single client. Some mechanisms [171, 101] assign queues to aggregated requests by their origin. Among them, Lee et al. [101] proposed a mechanism that provides differential guarantees to the aggregates based on the observation that bot distribution is not uniform across domains. However, queuing systems do not intend to offer nor can they provide precise waiting time guarantees to clients.

Gligor [67] proposed an alternate scheme that provides per-client, maximum waiting time guarantees via precisely time-scheduled service-access tokens. Such scheduling requires conservative workload prediction for every single service and assumes that all granted tokens would be used on time—which unavoidably leads to significant resource underutilization.

8.1.3 Fair Access

Another defense philosophy against DDoS attacks aims to offer a fair chance of resource access to users. This can be viewed as a last resort when it is difficult to differentiate legitimate traffic from malicious traffic.

Bandwidth control. Many of the bandwidth control mechanisms (especially fair queuing mechanisms) proposed to date can be used to prevent some (malicious) flows from exhausting the network bandwidth [112, 148, 111]. Pushback [112] focuses on rate limiting high bandwidth aggregates to their fair share of bandwidth, but it causes collateral damage as legitimate traffic in high bandwidth aggregates will be falsely throttled. CSFQ [148] achieves fair queuing without keeping per-flow state on core routers. NetFence [111] enables routers to signal congestion and guarantees per-sender fairness at bottleneck links.

In contrast to per-flow or per-sender fair bandwidth sharing, STRIDE enables endpoint domains and endhosts to negotiate their bandwidth shares, which makes it more practical considering the ISP business models.

To avoid granting access to nonexistent entities (e.g., via IP address spoofing) and to limit a client's attempt to gain advantage over others by masquerading as multiple entities, several DDoS defense mechanisms employ an interactive protocol that requires clients to present evidence proving their identity. Some of the mechanisms use a *proof of work* that a client is required to provide; others use a *credential* that proves their prior interaction.

Proof-of-work schemes.

- **Computation-based proof-of-work.** Many researchers have proposed computational puzzles [78, 43, 164, 129] demanding that clients to show their computational effort to get a service. Due to their simplicity and statelessness, computational puzzles are engaged in new network-layer as well as application-layer protocol designs. However, they cause high overhead to legitimate clients while providing only weak guarantees (i.e., weak probabilistic waiting time guarantee) [67], which has prevented them from being adopted in the real world.

- **Bandwidth-based proof-of-work.** Based on the observation that uplink bandwidth is another limited resource for clients, Walfish et al. [163] proposed to defeat DDoS attacks by encouraging clients to send a higher volume of traffic. *Legitimate* clients who have spare uplink bandwidth are expected to speak up, whereas *bad* clients (e.g., bots) cannot do so as they have already spoken up to flood the target. Though interesting, speak-up creates collateral damage by allowing legitimate clients to hurt each other. In RainCheck Filter, legitimate clients do not hurt each other.

- **Proof of human presence.** CAPTCHAs [161] use a hard artificial intelligence (AI) problem, which can be easily solved by most humans but not by machines (e.g., bots), to test for a human presence behind a service request. CAPTCHAs have been widely adopted by many web-based applications to test for a human presence and are also used to distinguish a flash crowd from a DDoS attack [79]. However, advances in CAPTCHA breaking techniques [169, 29, 28] weaken the effectiveness of this tool for DDoS defense. Furthermore, CAPTCHA's requirement for human interaction limits its applications.

- **Latency-based proof-of-work.** We follow a line of thought of latency-based proof-of-work [39, 111], where a server under a DoS attack prioritizes the requests of those clients who have waited longest for the service. Crowcroft et al. proposed a mechanism to enforce passive delay on clients, thus slowing down the request rate. This mechanism comes in two different forms: a centralized version using delay cookies and a distributed version requiring the client to contact several validators before being admitted by the server. However, in contrast to RainCheck Filter, this mechanism needs per-client state at the server and does not provide any service access guarantee.

Credentials. TCP SYN-cookie [18] is designed to allow a server to complete the TCP three-way handshake without keeping any connection state, such as the initial sequence number and the client address, that was exploited for flooding the server. The state is instead cryptographically encoded in a SYN cookie and provided to the client along with the server's SYN-ACK. The SYN cookie is then carried in the client's ACK so that the server can determine the legitimacy of the connection request.

Various proposals aiming for a faster web [135, 33, 46, 143, 141, 130] use a cryptographic credential (which is similar to a SYN cookie) to reduce the number of round trips for the connection establishment. TCP Fast Open (TFO) [135, 33] speeds up successive TCP connections using a TFO cookie, a server-generated Message Authentication Code (MAC) that proves the client's ownership of a source IP. TFO allows data to be sent before the completion of the three-way handshake with the help of TFO cookies.

Reusing prior session information would significantly improve the performance of secure protocols (such as SSL/TLS) by avoiding cryptographic parameter negotiation and expensive cryptographic operations for authentication and key setup. TLS protocol supports fast session resumption (i.e., abbreviated handshakes) using session IDs [46] or session tickets [143]. A session ID identifies the corresponding session state in the server's local cache, while a session ticket contains encrypted session state and thus avoids caching on the server. Quick UDP Internet Connection (QUIC) [141] and Minimal Latency Tunneling (MinimaLT) [130] are also designed to minimize connection setup latency. Both protocols can achieve one round trip for the very first connection and zero round trips (i.e., sending data packets right after client hello) for the following connections. In terms of DoS resilience, the current specification of QUIC eliminates DoS from spoofed IP addresses using ownership credentials. MinimaLT uses client puzzles to rate limit DoS attacks and avoids traffic amplification by ensuring that response packets are smaller than request packets.

Technically, RainCheck Filter creates credentials in a similar way to the aforementioned mechanisms. However, a key distinction is that each raincheck ticket contains a fine-grained timestamp by which the Raincheck Filter protocol performs admission control, guaranteeing a maximum waiting time for establishing a connection. Moreover, Raincheck Filter is a generic primitive that can mitigate server DoS, including SYN flooding.

8.2 Countermeasures to Selective Dropping

This section first reviews prior work on anonymous communication, which is the most common technique used to mitigate selective dropping based on network identifiers or topological locations. Note that standard end-to-end encryption techniques (e.g., IPsec) can be used to prevent discrimination based on content or higher-layer headers.

The most closely related schemes for anonymity protection, namely Tor Instead of IP [108] and AHP [136], are described and compared in LAP's security analysis section (Section 6.6).

Low-stretch anonymity systems. Using a single anonymous proxy such as anonymizer.com[2] results in low path stretch. However, users have to trust a remote proxy in burying the linkage between a sender and a receiver, and the proxy could easily become a single point of failure. uProxy[3] is a decentralized proxy service designed to circumvent Internet censorship, thereby avoiding the single point of failure problem. A user can connect to the Internet via a friend who acts as a proxy. However, the user has to place his or her full trust in this friend.

High-stretch anonymity systems. In Chaum's mix network [32], layer-encrypted messages are sent through a list of mixes, each of which can buffer, reorder, decrypt/encrypt these messages to defend against a global eavesdropper. However, this delaying and reordering renders this system impractical for real-time communication.

Onion routing systems, such as Tor [47], enable low-latency, bi-directional anonymous communication by sending layer-encrypted packets through indirect and unpredictable cryptographic circuits [153]. Unlike mix networks, onion routing systems are designed to defend against a local attacker (or a government-class attacker, as referred to in Chapter 6) that observes only a fraction of the network. In some realistic attacker scenarios, onion routing systems are shown to be more secure than mix networks [154]. Tarzan [65] explores onion routing in a peer-to-peer setting, and ANDaNA [45] adopts Tor in content-centric networking. However, onion routing systems still suffer from high latency due to high path stretch. To reduce Tor's latency, new relay selection algorithms are suggested to consider relay geolocations or link characteristics in addition to relay bandwidth [128, 145]. However, further studies are required to understand the impact of these algorithms on existing attacks against Tor.

Researchers have also explored solutions without layered encryption. For example, Information Slicing [84] achieves source and destination anonymity through multipath and secret sharing. However, Information Slicing operates on overlays and suffers from noticeable latency. Crowds [138] leverages a crowd of users to collaboratively remove the trace of the real requester, and Hordes [146] exploits the inherent crowds within muticast groups for receiver anonymity. However, both Crowds and Hordes significantly stretch end-to-end paths.

Detecting traffic discrimination. A long line of literature has focused on detecting traffic discrimination (or net neutrality violations) [155, 181, 49, 81]. These detection techniques in themselves are insufficient to mitigate traffic discrimination, especially when the sender has no choice of the communication path.

[2]`http://www.anonymizer.com/`
[3]`http://uproxy.org/`

Censorship resilience. Censorship-resilient systems such as Decoy routing [82], Telex [167], and Cirripede [76] rely on ISPs to redirect traffic to blocked destinations. Although they also require enlisting ISPs for protection as LAP does, they place trust in remote ISPs to help defend against a much stronger adversary who eavesdrops the local networks.

Attacks on anonymity systems. Several researchers have studied how to passively and actively attack anonymity systems. For passive attacks, the adversary attempts to de-anonymize traffic by observing side-channel information such as packet timing [106], clock skew [90], and unique system state [54, 172]. However, such passive attacks often fail to scale or rely on information leaked from higher layer protocols. On the other hand, active attacks can accelerate traffic correlation. DoS is one type of active attack that can be used for additional attack opportunities [21]. For example, by clogging the network and monitoring the latency change, the attacker can identify Tor entry nodes [121, 60] and locate Tor users [74]. Although our main objective is to camouflage one's topological location to enhance anonymity and privacy, LAP can also mitigate DoS-based attacks by selectively publishing encrypted paths.

Low-latency anonymity systems are shown to be inherently vulnerable to timing and traffic analysis [74, 31, 120] because an adversary can easily correlate the traffic patterns of a sender and a receiver. Since our goal in this dissertation is to provide topological anonymity, we consider such temporal side-channel attacks as future work.

8.3 Traffic Monitoring

Section 5.1.1 classifies prior work on detecting large flows and its closely related problem of finding frequent items based on the types of monitoring windows. Cormode and Hadjieleftheriou present a thorough survey and comparison of algorithms for finding frequent items [36]. This section reviews prior approaches based on their techniques. Most prior work does not consider the arbitrary window model.

Counter-based techniques. Counter-based techniques maintain a small number of counters, each of which is associated with a flow or an item. Manku and Motwani present another well-known counter-based technique called Lossy Counting [114]. For each stored item, Lossy Counting maintains and updates the upper bound and lower bound on the count of the item. The algorithm stores every new item and periodically removes items whose upper bound is less than the threshold. Similar to the MG algorithm discussed in Section 5.2, the Space Saving algorithm [116] proposed by Metwally et al. maintains k (item, counter) pairs and increases the corresponding counter of each incoming item. If the new item e is not stored currently, the stored item with the lowest count is replaced by the new item, and the counter increases accordingly.

Sketch-based techniques. Multistage filters identify large flows over fixed time windows [58] and over arbitrary windows [57]. Fang et al. [61] proposed a similar multistage algorithm but require more than one pass over the input stream. Cormode and Muthukrishnan present a novel data structure called count-min sketch, which summarizes an input stream and can answer queries such as finding frequent items [38]. As pointed out in their paper, despite the fact that the construction is similar to that of multistage filters, count-min sketches can flexibly support negative weights and require only pairwise independence hash functions rather than fully independent ones. In general, sketches can support a richer set of queries with a higher memory overhead compared with counter-based techniques.

Sampling-based techniques. Sampled NetFlow[4] maintains a generic traffic summary of sampled packets. With a sampling rate $1/r$, the frequency estimate is derived by multiplying the count by r. To improve the accuracy of the estimates, both Sticky sampling [114] and Sample-and-Hold [58] examine every incoming item and increase the corresponding count if the item is being monitored. If the new item is not being monitored, it is sampled and added to the monitoring list with a certain probability. Sampling-based techniques in general cannot achieve high accuracy due to the lack of per-packet information. Duffield [51] studies how to perform fair sampling in traffic flow measurements.

Other traffic summaries. A rich body of the network monitoring literature explores existing data streaming algorithms for other research topics such as flow counting [59], estimating flow distributions [92, 96], measuring flow matrices [184], identifying hierarchical heavy hitters [37, 183], and superspreader detection [160]. Muthukrishnan surveys data streaming algorithms and their applications, including network monitoring [122].

Calders et al. [30] define a new frequency measure as the maximum frequency over all possible windows ending at the current time. Although the idea of considering all possible windows is the same as the arbitrary window model, their algorithm focuses on accurately estimating the frequency based on the new frequency measure, whereas we seek to accurately identify large flows.

[4]Random Sampled NetFlow. `http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/nfstatsa.html`

8.4 Topology and Routing for High Availability

This dissertation concentrates on securing packet forwarding despite active adversaries. In addition to forwarding, researchers have also explored availability problems with respect to topology and routing.

8.4.1 Topology

Perfectly Secure Message Transmission (PSMT). The goal of Perfectly Secure Message Transmission (PSMT) is to achieve perfect secrecy and perfect resiliency in the presence of a computationally unbounded adversary who can corrupt up to a threshold number of nodes in the network and can behave arbitrarily. Work on PSMT focuses on deriving the theoretical bounds and topological characteristics for guaranteeing reliable and secure communication. For example, an early work [50] in this area proves that secure communication is possible under t corrupted nodes, if and only if the network is at least $2t + 1$ connected.

Because of the strong adversary model, PSMT proposals incur significant overhead in terms of computation, message size, and protocol rounds. Also, PSMT considers the communication between one sender and receiver pair, and thus cannot handle the case of resource competition between communication flows. This dissertation, which considers a weaker yet more practical adversary model, explores efficient security mechanisms for guaranteeing high availability when severe resource contention is possible. Moreover, communication can still be available even under low network connectivity.

Topological design. Research on topological resilience considers the problem of identifying critical links/nodes and adding redundancy to improve topological resilience against attacks [123, 95, 86, 87]. For example, Nagaraja and Anderson [123] consider node degree and betweenness centrality as connectivity metrics and adopt evolutionary game theory to analyze various topological defense mechanisms, such as replacing critical nodes (i.e., nodes with high degree or high betweenness centrality) with rings or cliques, or delegating connections of critical nodes to their neighbors. Rather than considering node degree or betweenness centrality, Kim et al. [86] consider a node to be critical if its removal disconnects the network. They identify critical nodes in ad hoc networks using the Laplacian matrix, and then improve topological resilience by adding backup links around the critical nodes.

8.4.2 Routing

Current approaches to enhance BGP stability fall into two categories. In the first category of research, researchers derive conditions that avoid conflict routing policies, thus guaranteeing BGP convergence [70, 159, 66, 147]. For example, the Gao-Rexford guideline suggests a class of policies based on business relationships (e.g., selecting routes from customers over routes from providers). Sobrinho [147] presents a routing algebra that can be applied to all path-vector routing protocols. Sobrinho shows that strict monotonicity of policies implies routing protocol convergence. With a monotone algebra, every network can be made free (e.g., no infinite loop) by breaking ties with the order (length) of the path or by avoiding domains that are providers (directly or indirectly) of one of its providers. Based on the algebra, FSR [139] presents an automatic toolkit to validate routing policies. However, such approaches assume that every AS complies with the suggested policies and thus cannot handle the instability caused by misconfigurations or adversarial behaviors.

In the second category of research, ASes deal with unstable or failed routes at runtime. For example, consensus routing [77] proposes adding a stable mode of packet delivery in which all routers on the path have to agree to use that path. However, there is no guarantee that ASes on a path can ever reach consensus. Additionally, malicious routers can prevent consensus to be reached. LIFEGUARD [85] proposes a practical repair of persistent route failures in which edge ISPs advertise carefully crafted routes that are prepended with the failed AS to trigger loop detection, thus avoiding the failed AS. Route Flap Damping, similarly, can mitigate persistent route flapping by suppressing unstable routes. However, such engineering hacks may introduce unforeseen side effects that exacerbate the stability problem in unexpected ways [175, 151].

Chapter 9

Conclusion and Future Work

A highly available Internet is critically important individuals, organizations, and nations. This dissertation explores how to provide availability guarantees despite active data-plane attacks on top of a future Internet architecture that supports top-down route discovery, isolation domains, and path control. As the number of future Internet projects initiated by both public and private sectors continues to increase, we expect to see more research endeavors to lower the barriers to deployment in the near future. For example, Raghavan et al. [137] propose a software-defined Internet architecture that aims to ease the task of adopting clean-slate Internet architectures.

This dissertation is an initial step toward a highly available Internet, and therefore focuses more on the conceptual design rather than on the actual implementation. While the overall solution is still in an early stage, several of the proposed ideas have been separately evaluated for their ability to improve availability even in the current Internet.

Improving Internet availability is a challenging problem because the current Internet is only available if all of its layers are available. Therefore, a complete solution for Internet availability must integrate techniques for all layers. This dissertation focuses on one challenging aspect of the problem: the mitigation of emerging data-plane attacks at the network layer that have not yet been effectively or efficiently resolved. Looking forward, we would like to explore solutions in multiple complementary dimensions with the hope of moving a step closer to a highly available Internet. We conclude this dissertation with future work summarized as follows.

- **Topology.** Multipath communication within specific topologies can create an asymmetry between attacker and defender. An interesting future direction is the study of how to construct network topologies for optimal availability guarantees while preventing attackers' misuse of multipath.

- **Routing.** Given a topology and policies, the route computation should derive at least one policy-compliant path between the source and the destination in a timely manner. An important line of future work would be the analysis of existing clean-slate routing architectures for provable guarantees and the formal examination of whether they can promptly converge to a consistent state after changes in topology or policy.

- **Forwarding.** Given one or multiple paths, the forwarding plane should ensure that the packets arrive at the destination with high probability. This dissertation addresses a part of this challenge and ensures timely packet delivery in the presence of address-based selective dropping and flooding attacks. In future work, we plan to investigate the impact of route and topology dynamics on forwarding availability.

Since improving Internet availability is more than just a technical problem, a natural long-term direction is the consideration of the policy and economic aspects of Internet availability. There is a particular need for the study of interdomain SLAs and business models and for an exploration into the possibility of availability-as-a-service model in which Internet users pay to secure their end-to-end communication with a certain level of availability, and receive compensations from providers that fail to satisfy their promises.

In summary, while high availability is required for many critical services that rely on the Internet, the current Internet was not designed with security and availability in mind. In this dissertation, we designed security mechanisms for achieving availability guarantees on top of a clean-slate architecture despite active data-plane attacks. In particular, we design efficient defense mechanisms that help set an upper bound on the waiting time and a lower bound on the reservable bandwidth in the presence of link-flooding DDoS and selective dropping. Prior work cannot achieve these same guarantees. Our results demonstrate how much better the Internet could be, reveal issues and possibilities, and help plan the trajectory of evolution. We hope this work can represent a first step toward a highly available Internet where users can access Internet-based services whenever they want, especially in critical situations.

Bibliography

[1] CAIDA: The Cooperative Association for Internet Data Analysis. http://www.caida.org/. 36, 38, 42

[2] MobilityFirst Future Internet Architecture Project. http://mobilityfirst.winlab.rutgers.edu/. 6, 100, 112, 117

[3] New Study Reveals the Impact of Travel Site Performance on Consumers. http://www.akamai.com/html/about/press/releases/2010/press_061410.html. 5, 45

[4] SJCL PBKDF2 Benchmark. https://wiki.mozilla.org/SJCL_PBKDF2_Benchmark. 48

[5] Social, Digital & Mobile in 2014. http://wearesocial.sg/blog/2014/01/social-digital-mobile-2014/. 1

[6] SPDY: An Experimental Protocol for a Faster Web. http://dev.chromium.org/spdy/spdy-whitepaper. 57

[7] The CAIDA UCSD Anonymized Internet Traces 2012. http://www.caida.org/data/passive/passive_2012_dataset.xml. 88

[8] The RouteViews Project. http://www.routeviews.org. 111, 112

[9] Traces 1 of TCP Port 80 Traffic Traces from Federico II. http://traffic.comics.unina.it/Traces/ttraces.php. 88

[10] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proceedings of ACM SIGCOMM*, 2008. 70

[11] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, W. Lehr, B. T. Loo, D. Mazieres, A. Nicolosi, J. M. Smith, I. Stoica, R. van Renesse, M. Walfish, H. Weatherspoon, and C. S. Yoo. The NEBULA Future Internet Architecture. In *The Future Internet*, pages 16–26. Springer, 2013. 6

[12] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. *ACM SIGCOMM Computer Communication Review*, 34(1):39–44, 2004. 18, 22

[13] A. Arasu and G. S. Manku. Approximate Counts and Quantiles over Sliding Windows. In *Proceedings of ACM PODS*, 2004. 71

[14] K. Argyraki and D. R. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Proceedings of USENIX ATEC*, 2005. 146

[15] K. Argyraki and D. R. Cheriton. Network Capabilities: The Good, the Bad and the Ugly. In *Proceedings of ACM HotNets*, 2005. 18, 146

[16] I. Avramopoulos and J. Rexford. Stealth Probing: Efficient Data-Plane Security for IP Routing. In *Proceedings of USENIX ATC*, 2006. 48, 98

[17] I. Avramopoulos, J. Rexford, D. Syrivelis, and S. Lalis. Counteracting Discrimination against Network Traffic. Technical report, TR-794-07, Princeton University Computer Science, 2007. 98

[18] D. J. Bernstein. SYN cookies. `http://cr.yp.to/syncookies.html`, 1996. 148

[19] O. Berthold, A. Pfitzmann, and R. Standtke. The Disadvantages of Free MIX Routes and How to Overcome Them. In *Proceedings of PETS*, 2001. 101

[20] F. Bonomi and K. Fendick. The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service. In *IEEE Network Magazine, vol. 9, no. 2*, pages 25–39, 1995. 146

[21] N. Borisov, G. Danezis, P. Mittal, and T. Parisa. Denial of Service or Denial of Security? How Attacks on Reliability can Compromise Anonymity. In *Proceedings of ACM CCS*, 2007. 123, 151

[22] B. Boyer and J. Moore. A Fast Majority Vote Algorithm. Technical report, ICSCA-CMP-32, Institute for Computer Science, University of Texas, 1981. 77

[23] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), Sept. 1997. Updated by RFCs 2750, 3936. 18, 146

[24] R. J. Branaghan and C. A. Sanchez. Feedback Preferences and Impressions of Waiting. *Journal of the Human Factors*, 51(4):528–538, 2009. 5, 45, 58, 60

[25] P. Bright. Verizon could be throttling Netflix and Amazon, but there's no actual evidence of it. `http://arstechnica.com/information-technology/2014/02/verizon-could-be-throttling-netflix-and-amazon-but-theres-no-actual-evidence-of-it/`, 2014. 4

[26] H. Burch and B. Cheswick. Tracing Anonymous Packets to Their Approximate Source. In *Proceedings of LISA*, 2000. 123

[27] S. Burnett, N. Feamster, and S. Vempala. Chipping Away at Censorship Firewalls with User-Generated Content. In *Proceedings of USENIX Security*, 2010. 103

[28] E. Bursztein, R. Beauxis, H. Paskov, D. Perito, C. Fabry, and J. Mitchell. The Failure of Noise-Based Non-continuous Audio Captchas. In *Proceedings of IEEE Symposium on Security and Privacy*, 2011. 58, 148

[29] E. Bursztein, M. Martin, and J. Mitchell. Text-based CAPTCHA Strengths and Weaknesses. In *Proceedings of ACM CCS*, 2011. 58, 148

[30] T. Calders, N. Dexters, and B. Goethals. Mining Frequent Items in a Stream Using Flexible Windows. *Intelligent Data Analysis*, 12(3):293–304, 2008. 152

[31] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic Analysis Against Low-Latency Anonymity Networks Using Available Bandwidth Estimation. In *Proceedings of ESORICS*, 2010. 103, 123, 151

[32] D. L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–90, Feb. 1981. 99, 150

[33] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. `http://tools.ietf.org/html/draft-ietf-tcpm-fastopen-05`, Retrieved Oct. 2013. 149

[34] K. Clay. Amazon.com Goes Down, Loses $66,240 Per Minute. `http://www.forbes.com/sites/kellyclay/2013/08/19/amazon-com-goes-down-loses-66240-per-minute/`. 1

[35] S. Cohen and Y. Matias. Spectral Bloom Filters. In *Proceedings of ACM SIGMOD*, 2003. 53

[36] G. Cormode and M. Hadjieleftheriou. Finding Frequent Items in Data Streams. *Proc. VLDB Endow.*, 1(2):1530–1541, 2008. 151

[37] G. Cormode, F. Korn, S. Muthukrishnan, and D. Sirvastava. Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-Dimensional Data. In *Proceedings of ACM SIGMOD*, 2004. 152

[38] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(1):58–75, 2005. 71, 74, 76, 152

[39] J. Crowcroft, T. Deegan, C. Kreibich, R. Mortier, and N. Weaver. Lazy Susan: Dumb Waiting as Proof of Work. Technical Report 703, University of Cambridge, UCAM-CL-TR-703, 2007. 148

[40] A. Dainotti, A. Pescapè, P. Salvo Rossi, F. Palmieri, and G. Ventre. Internet Traffic Modeling by means of Hidden Markov Models. *Computer Networks (Elsevier)*, 52:2645–2662, 2008. 88

[41] A. Dainotti, A. Pescapè, and G. Ventre. A Cascade Architecture for DoS attacks Detection based on the Wavelet Transform. *Journal of Computer Security*, 17(6/2009):945–968, 2009. 88

[42] D. Danchev. How much does it cost to buy 10,000 U.S.-based malware-infected hosts? http://www.webroot.com/blog/2013/02/28/how-much-does-it-cost-to-buy-10000-u-s-based-malware-infected-hosts/. 45

[43] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *Proceedings of USENIX Security*, 2001. 148

[44] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of ESA*, 2002. 71, 74, 76, 77

[45] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun. ANDaNA: Anonymous named data networking application. In *Proceedings of NDSS*, 2012. 150

[46] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008. 149

[47] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of conference on USENIX Security Symposium*, 2004. 98, 99, 103, 110, 119, 120, 121, 122, 150

[48] R. Dingledine and S. J. Murdoch. Performance improvements on Tor—or, why Tor is slow and what we're going to do about it. `https://www.torproject.org/press/presskit/2009-03-11-performance.pdf`, 2009. 99

[49] M. Dischinger, M. Marcon, S. Guha, K. P. Gummadi, R. Mahajan, and S. Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *Proceedings of USENIX NSDI*, 2010. 48, 150

[50] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly Secure Message Transmission. *Journal of the ACM (JACM)*, 1993. 153

[51] N. Duffield. Fair Sampling Across Network Flow Measurements. In *Proceedings of ACM SIGMETRICS*, 2012. 152

[52] J. Duffy. Cisco routers caused major outage in Japan: report. `http://www.networkworld.com/news/2007/051607-cisco-routers-major-outage-japan.html`, 2007. 3

[53] J. Duffy. Juniper at the root of Internet outage? `http://www.networkworld.com/news/2011/110711-internet-outage-252851.html`, 2011. 3

[54] P. Eckersley. How Unique Is Your Web Browser ? In *Proceedings of PETS*, 2010. 151

[55] A. Efrati. 'Like' Button Follows Web Users. `http://online.wsj.com/article/SB10001424052748704281504576329441432995616.html`, 2011. 101

[56] T. Elahi and I. Goldberg. CORDON–A Taxonomy of Internet Censorship Resistance Strategies. Technical report, CACR 2012-33, University of Waterloo, 2012. 98

[57] C. Estan. *Internet Traffic Measurement: What's Going on in my Network?* PhD thesis, 2003. 70, 71, 82, 86, 152

[58] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003. 70, 71, 74, 76, 82, 86, 152

[59] C. Estan, G. Varghese, and M. Fisk. Bitmap Algorithms for Counting Active Flows on High Speed Links. In *Proceedings of ACM IMC*, 2003. 152

[60] N. S. Evans, R. Dingledine, and C. Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *Proceedings of USENIX Security*, 2009. 103, 123, 151

[61] M. Fang and N. Shivakumar. Computing Iceberg Queries Efficiently. In *Proceedings of VLDB*, 1999. 71, 74, 152

[62] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704. 70

[63] M. Fischer and S. Salzberg. Finding a Majority Among N Votes: Solution to Problem 81-5. *Journal of Algorithms - JAL*, 3(4):362–380, 1982. 77

[64] P. Flajolet and G. N. Martin. Probabilistic Counting. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 1983. 60

[65] M. J. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer Michael. In *Proceedings of ACM CCS*, 2002. 150

[66] L. Gao and J. Rexford. Stable Internet Routing Without Global Coordination. *IEEE/ACM Transactions on Networking (TON)*, 9(6):681–692, 2001. 153

[67] V. D. Gligor. Guaranteeing Access in Spite of Distributed Service-Flooding Attacks. In *Proceedings of Security Protocols Workshop*, 2005. 46, 47, 147, 148

[68] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying Frequent Items in Sliding Windows over On-Line Packet Streams. In *Proceedings of ACM IMC*, 2003. 71

[69] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-Quality Monitoring in the Presence of Adversaries. In *Proceedings of ACM SIGMETRICS*, 2008. 48

[70] T. G. Griffin and G. Wilfong. An Analysis of BGP Convergence Properties. In *Proceedings of ACM SIGCOMM*, 1999. 153

[71] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources. In *Proceedings of IEEE ICNP*, 2004. 69, 88

[72] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. XIA: Efficient support for evolvable internetworking. In *Proceedings of USENIX NSDI*, 2013. 6

[73] J. Heidemann, L. Quan, and Y. Pradkin. A Preliminary Analysis of Network Outages During Hurricane Sandy. Technical Report November 2012, USC/ISI ISI-TR-685b, 2012. 2

[74] N. Hopper, E. Y. Vasserman, and E. Chan-TIN. How Much Anonymity does Network Latency Leak? *ACM Transactions on Information and System Security*, 13(2):1–28, Feb. 2010. 103, 151

[75] A. Houmansadr and N. Borisov. SWIRL: A Scalable Watermark to Detect Correlated Network Flows. In *Proceedings of NDSS*, 2011. 123

[76] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability. *Proceedings of ACM CCS*, 2011. 151

[77] J. P. John, E. Katz-bassett, A. Krishnamurthy, and T. Anderson. Consensus Routing: The Internet as a Distributed System. In *Proceedings of USENIX NSDI*, 2008. 154

[78] A. Juels and J. G. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of NDSS*, 1999. 148

[79] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *Proceedings of NSDI*, 2005. 148

[80] M. S. Kang, S. B. Lee, and V. D. Gligor. The Crossfire Attack. In *Proceedings of IEEE Symposium on Security and Privacy*, 2013. 145

[81] P. Kanuparthy and C. Dovrolis. DiffProbe: Detecting ISP Service Discrimination. In *Proceedings IEEE INFOCOM*, 2010. 150

[82] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy Routing: Toward Unblockable Internet Communication. *Proceedings of USENIX Workshop on Free and Open Communications on the Internet*, 2011. 151

[83] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Transactions on Database Systems*, 28(1):51–55, 2003. 71, 74, 76, 77

[84] S. Katti, J. Cohen, and D. Katabi. Information Slicing: Anonymity Using Unreliable Overlays. In *Proceedings of USENIX NSDI*, 2007. 150

[85] E. Katz-Bassett, C. Scott, D. R. Choffnes, I. Cunha, V. Valancius, N. Feamster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy. LIFEGUARD: Practical Repair of Persistent Route Failures. In *Proceedings of ACM SIGCOMM*, pages 395–406, 2012. 154

[86] T. Kim, D. Tipper, P. Krishnamurthy, and A. L. Swindlhurst. Improving the Topological Resilience of Mobile Ad Hoc Networks. In *Proceedings of DRCN*, 2009. 153

[87] T.-H. Kim, D. Tipper, and P. Krishnamurthy. Improving the Connectivity of Heterogeneous Multi-Hop Wireless Networks. In *Proceedings of IEEE ICC*, 2011. 153

[88] R. Kohavi and R. Longbotham. Online Experiments: Lessons Learned. *Computer*, 40(9):103–105, 2007. 1, 99

[89] E. Kohler, R. Morris, B. Chen, and J. Jannotti. The Click Modular Router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000. 37

[90] T. Kohno, A. Broido, and K. Claffy. Remote Physical Device Fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, Feb. 2005. 123, 151

[91] J. Krumm. A Survey of Computational Location Privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, 2009. 102

[92] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution. In *Proceedings of ACM SIGMETRICS*, 2004. 152

[93] A. Kuzmanovic and E. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks and Counter Strategies. *IEEE/ACM Transactions on Networking*, 14(4):683–696, 2006. 69, 88, 92

[94] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proceedings of ACM SIGCOMM*, 2000. 3

[95] M. Lad, R. Oliveira, B. Zhang, and L. Zhang. Understanding Resiliency of Internet Topology Against Prefix Hijack Attacks. *Proceedings of IEEE/IFIP DSN*, 2007. 153

[96] A. Lall, V. Sekar, M. Ogihara, J. J. Xu, and H. Zhang. Data Streaming Algorithms for Estimating Entropy of Network Traffic. In *Proceedings of ACM SIGMETRICS*, 2006. 152

[97] S. LaPerrière. Taiwan Earthquake Fiber Cuts: a Service Provider View, 2007. 2

[98] G. S. Lee and B. Thuraisingham. Cyberphysical Systems Security Applied to Telesurgical Robotics. *Computer Standards & Interfaces*, 34(1):225–229, Jan. 2012. 2

[99] L. Lee and H. Ting. A Simpler and More Efficient Deterministic Scheme for Finding Frequent Items over Sliding Windows. In *Proceedings of ACM PODS*, 2006. 71

[100] S. Lee and V. Gligor. FLoc: Dependable Link Access for Legitimate Traffic in Flooding Attacks. *Proceedings of IEEE ICDCS*, 2010. 146

[101] S. B. Lee, V. D. Gligor, and A. Perrig. Dependable Connection Setup for Network Capabilities. In *Proceedings of IEEE DSN*, 2010. 147

[102] S. B. Lee, M. S. Kang, and V. D. Gligor. CoDef: Collaborative Defense Against Large-Scale Link-Flooding Attacks. In *Proceedings of ACM CoNEXT*, 2013. 48, 58, 144

[103] Leichtman Research Group. Nearly 1.3 Million Add Broadband In The First Quarter of 2011. `http://www.leichtmanresearch.com/press/051711release.pdf`, 2011. 112

[104] J. Lemon. Resisting SYN Flood DoS Attacks with a SYN Cache. In *BSDCon*, 2002. 47

[105] P. G. Leon, B. Ur, R. Balebako, L. F. Cranor, R. Shay, and Y. Wang. Why Johnny Can't Opt Out: A Usability Evaluation of Tools to Limit Online Behavioral Advertising. In *Proceedings of CHI*, 2012. 99

[106] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. Timing Attacks in Low-Latency Mix Systems. In *Proceedings of Financial Cryptography*, 2004. 151

[107] Z. Li, A. Goyal, and Y. Chen. Honeynet-based Botnet Scan Traffic Analysis. *Botnet Detection*, 36:25–44, 2008. 58

[108] V. Liu, S. Han, A. Krishnamurthy, and T. Anderson. Tor instead of IP. In *Proceedings of ACM HotNets*, 2011. 101, 119, 120, 121, 122, 149

[109] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *Proceedings of USENIX/ACM NSDI*, 2008. 70

[110] X. Liu, X. Yang, and Y. Lu. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. In *Proceedings of ACM SIGCOMM*, number 4, 2008. 146

[111] X. Liu, X. Yang, and Y. Xia. NetFence: Preventing Internet Denial of Service from Inside Out. In *Proceedings of ACM SIGCOMM*, 2010. 48, 147, 148

[112] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, July 2002. 147

[113] D. H. Maister. *The Psychology of Waiting Lines*. Harvard Business School, 1984. 5, 45

[114] G. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of VLDB*, 2002. 71, 74, 76, 151, 152

[115] J. Markoff and N. Perlroth. Firm Is Accused of Sending Spam, and Fight Jams Internet. `http://www.nytimes.com/2013/03/27/technology/internet/online-dispute-becomes-internet-snarling-attack.html`, 2013. 4

[116] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proceedings of ICDT*, 2005. 71, 74, 76, 151

[117] J. Misra and D. Gries. Finding Repeated Elements. *Science of Computer Programming*, 2(2):143–152, 1982. 71, 74, 76

[118] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. Stealthy Traffic Analysis of Low-Latency Anonymous Communication Using Throughput Fingerprinting. In *Proceedings of ACM CCS*, 2011. 123

[119] J. A. Muir and P. C. V. Oorschot. Internet Geolocation: Evasion and Counterevasion. *ACM Comput. Surv.*, 42:4:1–4:23, December 2009. 98

[120] S. Murdoch. Hot or Not: Revealing Hidden Services by their Clock Skew. In *Proceedings of ACM CCS*, 2006. 151

[121] S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Proceedings of IEEE Symposium on Security and Privacy*, 2005. 103, 123, 151

[122] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005. 152

[123] S. Nagaraja and R. Anderson. The Topology of Covert Conflict. In *Proceedings of Workshop on Economics of Information Security*, 2006. 153

[124] F. F.-H. Nah. A Study on Tolerable Waiting Time: How Long are Web Users Willing to Wait? *Behaviour & Information Technology*, 23(3):153–163, May 2004. 5, 45, 58, 60

[125] M. Naor and E. Yogev. Tight Bounds for Sliding Bloom Filters. *pre-print*, pages 1–18, 2013. 53

[126] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra. Verifying and Enforcing Network Paths with ICING. In *Proceedings of ACM CoNext*, 2011. 70, 100, 129

[127] H. Noman and J. C. York. West Censoring East: The Use of Western Technologies by Middle East Censors, 2010-2011. `https://opennet.net/west-censoring-east-the-use-western-technologies-middle-east-censors-2010-2011`. 4

[128] A. Panchenko, L. Pimenidis, and J. Renner. Performance Analysis of Anonymous Communication Channels Provided by Tor. In *Proceedings of International Conference on Availability, Reliability and Security*, 2008. 99, 150

[129] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *Proceedings of ACM SIGCOMM*, 2007. 46, 146, 148

[130] W. Petullo, X. Zhang, J. Bernstein, and T. Lange. MinimaLT: Minimal-latency Networking Through Better Security. In *Proceedings of ACM CCS*, 2013. 149

[131] A. Pfitzmann and K. Marit. Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology. In *Proceedings of PETS*, 2001. 101

[132] Ponemon Institute. Cyber Security on the Offense: A Study of IT Security Experts, 2012. 45

[133] A. Popescu, T. Underwood, and E. Zmijewski. Quaking Tables: The Taiwan Earthquakes and the Internet Routing Table, 2007. 2

[134] Prolexic. Prolexic Attack Report Q1 2012, 2012. 4

[135] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. TCP Fast Open. In *Proceedings of ACM CoNEXT*, 2011. 149

[136] B. Raghavan, T. Kohno, A. C. Snoeren, and D. Wetherall. Enlisting ISPs to Improve Online Privacy: IP Address Mixing by Default. In *Proceedings of PETS*, 2009. 99, 101, 111, 119, 120, 121, 149

[137] B. Raghavan, T. Koponen, A. Ghodsi, M. Casado, S. Ratnasamy, and S. Shenker. Software-Defined Internet Architecture: Decoupling Architecture from Infrastructure. In *Proceedings of ACM HotNets*, 2012. 12, 155

[138] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998. 150

[139] Y. Ren, W. Zhou, A. Wang, L. Jia, A. J. Gurney, B. T. Loo, and J. Rexford. FSR: Formal Analysis and Implementation Toolkit for Safe Inter-domain Routing. *ACM SIGCOMM Computer Communication Review*, 41(4):440–441, 2011. 154

[140] Reporters Without Borders. Enemies of the Internet. Special Edition: Surveillance, 2013. 4

[141] J. Roskind. QUIC: Design Document and Specification Rational. `https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34`, Retrieved Oct. 2013. 149

[142] S. Russolillo. After Nasdaq Halt, Watch Apple's Trading Volume Vanish. `http://blogs.wsj.com/moneybeat/2013/08/22/after-nasdaq-halt-watch-apples-trading-volume-vanish/`, 2013. 1

[143] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077, Jan. 2008. 149

[144] W. Scott, R. Cheng, J. Li, A. Krishnamurthy, and T. Anderson. Blocking-Resistant Network Services using Unblock, 2012. 98

[145] M. Sherr, M. Blaze, and B. T. Loo. Scalable Link-Based Relay Selection for Anonymous Routing. In *Proceedings of PETS*, 2009. 150

[146] C. Shields and B. N. Levine. A Protocol for Anonymous Communication Over the Internet. In *Proceedings of ACM CCS*, 2000. 150

[147] J. L. Sobrinho. An Algebraic Theory of Dynamic Network Routing. *IEEE/ACM Transactions on Networking (TON)*, 13(5):1160–1173, 2005. 153, 154

[148] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks. *IEEE/ACM Transactions on Networking*, 11(1):33–46, Feb. 2003. 22, 147

[149] A. Studer and A. Perrig. The Coremelt Attack. In *Proceedings of ESORICS*, 2009. 6, 17, 19, 145

[150] A. Studer, E. Shi, F. Bai, and A. Perrig. TACKing Together Efficient Authentication, Revocation, and Privacy in VANETs. In *In Proceedings of IEEE SCEON*, 2009. 139

[151] M. Suchara, A. Fabrikant, and J. Rexford. BGP Safety with Spurious Updates Martin. In *Proceedings of IEEE INFOCOM*, 2011. 154

[152] P. Syverson. Why I'm not an Entropist. In *International Workshop on Security Protocols*. Springer-Verlag, LNCS, 2009. 101

[153] P. Syverson. A Peel of Onion. In *Proceedings of ACSAC*, 2011. 150

[154] P. Syverson. Sleeping dogs lie in a bed of onions but wake when mixed. In *Proceedings of HotPETs*, 2011. 150

[155] M. B. Tariq, M. Motiwala, N. Feamster, and M. Ammar. Detecting Network Neutrality Violations with Causal Inference. In *Proceedings of ACM CoNext*, 2009. 150

[156] The European Union Agency for Network and Information Security (ENISA). Annual Incident Reports 2012, 2013. 2

[157] J. Trostle, H. Matsuoka, J. Kempf, T. Kawahara, and R. Jain. Cryptographically Protected Prefixes for Location Privacy in IPv6. In *Proceedings of PETS*, 2004. 121

[158] D. M. Turner, V. Prevelakis, and A. D. Keromytis. A Market-Based Bandwidth Charging Framework. *ACM Transactions on Internet Technology*, 10(1):1–30, 2010. 92

[159] K. Varadhan, R. Govindan, and D. Estrin. Persistent Route Oscillations in Inter-domain Routing. *Computer Networks*, 32(1):1–16, Jan. 2000. 153

[160] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum. New Streaming Algorithms for Fast Detection of Superspreaders. In *Proceedings of NDSS*, 2005. 152

[161] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology–EUROCRYPT*, 2003. 58, 148

[162] A. Vulimiri, G. a. Agha, P. B. Godfrey, and K. Lakshminarayanan. How Well Can Congestion Pricing Neutralize Denial of Service Attacks? In *Proceedings of ACM SIGMETRICS*, 2012. 28

[163] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS Defense by Offense. In *Proceedings of ACM SIGCOMM*, 2006. 46, 148

[164] X. Wang and M. K. Reiter. Defending Against Denial-of-Service Attacks with Puzzle Auction. In *Proceedings of IEEE Symposium on Security and Privacy*. 63, 148

[165] Z. Wang, Z. Qian, Q. Xu, Z. M. Mao, and M. Zhang. An Untold Story of Middleboxes in Cellular Networks. *ACM SIGCOMM Computer Communication Review*, 41(4):374–385, 2011. 111

[166] World Economic Forum. Global Risks 2014 Ninth Edition, 2014. 1

[167] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the Network Infrastructure. *Proceedings of USENIX Security*, 2011. 151

[168] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *Proceedings of IEEE Symposium on Security and Privacy*, 2004. 18, 22, 146

[169] J. Yan and A. S. El Ahmad. A Low-cost Attack on a Microsoft Captcha. In *Proceedings of ACM CCS*, 2008. 58, 148

[170] X. Yang, D. Clark, and A. W. Berger. NIRA: A New Inter-Domain Routing Architecture. *IEEE/ACM Transactions on Networking*, 15(4):775–788, Aug. 2007. 20

[171] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-Limiting Network Architecture. *IEEE/ACM Transactions on Networking*, 16(6):1267–1280, Dec. 2008. 22, 146, 147

[172] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi. Host Fingerprinting and Tracking on theWeb: Privacy and Security Implications. In *Proceedings of NDSS*, 2012. 151

[173] B. Yener, Y. Ofek, and M. Yung. Combinatorial Design of Congestion-Free Networks. *IEEE/ACM Transactions on Networking (TON)*, 5(6):989–1000, 1997. 19

[174] M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of USENIX NSDI*, 2013. 81

[175] B. Zhang, D. Pei, D. Massey, and L. Zhang. Timer Interaction in Route Flap Damping. In *Proceedings of IEEE ICDCS*, 2005. 154

[176] L. Zhang, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named Data Networking. Technical report, NDN-0019, 2014. 6

[177] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVartion Protocol. *Network, IEEE*, 7(September):8–18, 1993. 5, 41

[178] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. SCION: Scalability, Control, and Isolation on Next-Generation Networks. In *Proceedings of IEEE Symposium on Security and Privacy*, 2011. 6, 7, 12, 14, 18, 20, 100, 112, 115

[179] X. Zhang, C. Lan, and A. Perrig. Secure and Scalable Fault Localization under Dynamic Traffic Patterns. In *Proceedings of IEEE Security and Privacy*, 2012. 48, 129

[180] X. Zhang, Z. Zhou, H.-C. Hsiao, T. H.-J. Kim, A. Perrig, and P. Tague. ShortMAC: Efficient Data-Plane Fault Localization. In *Proceedings of NDSS*, 2012. 40

[181] Y. Zhang, Z. Mao, and M. Zhang. Detecting Traffic Differentiation in Backbone ISPs with NetPolice. *Proceedings of ACM IMC*, 2009. 150

[182] Y. Zhang, Z. M. Mao, and M. Zhang. Ascertaining the Reality of Network Neutrality Violation in Backbone ISPs. *Proceedings of ACM HotNets*, 2008. 4

[183] Y. Zhang, S. Singh, and S. Sen. Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications. In *Proceedings of ACM IMC*, 2004. 152

[184] Q. G. Zhao, A. Kumar, J. Wang, and J. J. Xu. Data Streaming Algorithms for Accurate and Efficient Measurement of Traffic and Flow Matrices. In *Proceedings of ACM SIGMETRICS*, 2005. 152