Cooperative Network Systems for Protecting Internet Users

Sophia S. Yoo

A dissertation
presented to the faculty
of Princeton University
in candidacy for the degree
of Doctor of Philosophy

recommended for acceptance
by the Department of
Electrical and Computer Engineering

Advisers: Professor Jennifer Rexford, Professor Maria Apostolaki

May 2026

Abstract

Despite decades of evolution, the modern Internet still suffers from architectural security issues, which leave users exposed to persistent threats undermining availability, privacy, and performance. At the heart of these vulnerabilities are two fundamentally limited forms of control: *ingress control* to mitigate large-scale volumetric attacks, and *route control* to steer traffic away from untrusted or degraded interdomain paths. We observe that the *network edge* is a natural point of leverage for addressing both problems, but note that its power is inherently limited when exercised alone. A *single* edge platform cannot enforce ingress defenses at the scale and sophistication necessitated by modern attacks, and a *single* edge domain cannot shape routing decisions beyond its direct upstream providers. Our key insight is that these limitations can be overcome by *distributing control through structured coordination*. However, this approach introduces a central systems challenge: once functionality is distributed, constituent components must still operate as a unified defense, despite heterogeneous platform constraints and untrusted intermediate networks.

We develop a set of systems that addresses these challenges, by progressively extending edge-based protection from coordinated defenses *within* a single edge domain to selective cooperation *between* domains. SmartCookie introduces a *layered co-design* for ingress control within an edge domain, partitioning defense logic between programmable network hardware for secure, high-speed filtering and downstream software for richer, more expressive policy enforcement. This co-design allows defenses to withstand attack-scale traffic without sacrificing performance. Tango extends the same principle across edge domains, enabling *selectively cooperating* edge networks to expose extra interdomain routing options while remaining BGP-compliant. Building on this, PraxiGuard models the privacy risks and performance costs of newly exposed routes and determines an optimal traffic allocation that balances operator-defined privacy and performance objectives. Across these systems, we develop lightweight *coordination protocols* and in-network *cryptographic primitives* that allow distributed components to exchange enforcement signals securely, and to operate cohesively, without relying on support from non-participating networks. Together, our contributions show that edge-centered co-design and cooperation offer a practical, principled way to strengthen ingress control and route control to better protect Internet users.

Contents

List of Figures

List of Tables

For my family—all of them.

Acknowledgments

I owe a debt of gratitude to my advisor, Jennifer Rexford, that no acknowledgment could ever fully capture. She took me on as a newcomer to networks and has been unwaveringly invested in my growth from day one. Her ability to gracefully balance rigorous technical detail with big-picture insights—all while communicating with unmatched clarity and a rare blend of genuine curiosity and human connection—has not only defined my doctoral experience but remains my enduring academic inspiration. Working with her has been an immense privilege and gift, profoundly shaping the researcher I am today.

I am equally indebted to my advisor, Maria Apostolaki, who stepped in midway through my Ph.D. and transformed its trajectory. I owe the focus and depth of my work to her uncompromising intellectual rigor. When personal challenges in my fifth year affected my professional capacity, Maria met every hurdle with compassion and resolve, helping me see my work as, above all, an investment in my own growth. She has been a relentless advocate, always believing in the best possible version of me and helping me pursue my research with clarity, precision, and courage. I could not have done this without her.

I am deeply grateful to the members of my committee—David Hay-Saikevich, Prateek Mittal, and Adrian Perrig—for their sustained guidance and support. I am especially grateful to David, who has been as much a collaborator and friend as a mentor; his intellectual generosity broadened my research in unexpected directions and made the work itself a genuine joy. I thank Prateek for his steady belief in my work since my early years and for the thoughtful, invaluable mentorship he offered during my academic search. I also thank Adrian for his incisive perspective and for his engaged stewardship of this work.

I have been fortunate to belong to not one, but two, extraordinary research groups, which quickly became my academic family. I could not have asked for better companions in the trenches than my Cabernet family, past and present. My gratitude goes to Sata Sengupta, for six years of everything from hours-long philosophical deep dives to close technical collaboration, making him as much a dear friend as a trusted colleague; to Danny Chen, for his bold, unapologetic energy, boundless curiosity, and the animated discussions that guided me from compiling my first Tofino program to filing my first patent; to Mary Hogan, the first person I met in the lab, for her steady, cheerful presence and invaluable support over the years, and especially during my academic search; to Robert MacDavid, for his infectious warmth and humor that brightened my early years; and to Yufei Zheng and Mengying Pan, for the keen insight and quiet depth they brought to all our years together.

My deepest thanks also go to John Sonchack, my personal P4/Tofino hero and perhaps the sharpest mind I have ever encountered, and to Oliver Michel, whose meticulous brilliance I hope to emulate; both came to know my work as closely as co-authors, reading drafts and listening to practice talks countless times, and offering a level of careful feedback and encouragement that made the work what it is today. I am also grateful to Liang Wang and Joon Kim for the knowledge and perspectives they generously shared with me.

Later in my Ph.D., I found a second home with my NetΣyn family. I am grateful to Fengchen Gong and Ann Zhou, for their support, friendship, and (in Ann's case) our shared love of cats; to Kostas Doumanidis, Minhao Jin, and Hongyu Hè for their inspiring energy and depth; and to Cleef, NetΣyn's spirited four-pawed collaborator, for adding his voice—literally—to many research discussions and never failing to make me smile.

Beyond the lab, many others shaped this work through their expertise and support. I thank Henry Birge-Lee for his collaboration across multiple projects; Alan Liu for his help in benchmarking the first part of this work; Benji Herber for the energy and fun he brought to our collaborations; Sofia Marina for the exceptional creativity and dedication she brought to our work together; Joe Karam for sharing his expertise and for being such a pleasure to work with; and Julie Yun, who saw something in me when I was still an undergraduate junior and has cheered me on through every milestone since. Thank you also to Dr. J, for all our many talks: you really made a difference in my life.

To my undergraduate friends, whose presence has remained a beautiful constant well beyond those early years: I am deeply grateful for all the ways you continued to show up, support me, and shape my graduate journey. To Melissa, Sarah, Emma, and Haley—my personal cheering squad: thank you for walking this chapter with me, for reminding me to eat and rest, and for always picking up the phone to share hours-long conversations while rooting for me without hesitation. To Valeria—a kindred spirit and my constant sounding board for both the personal and the intellectual: thank you for the countless texts, the philosophical exchanges, and your sharp perspective, especially in our shared "hot takes." And to Julie, Jackie, Chitra, Amanda, and Chaereen: thank you for staying connected across every distance, always believing in me, and making life feel full whenever we meet.

One of the most special parts of graduate school was being surrounded by people who are not only brilliant, but who bring a rare and thoughtful depth to everything they touch. To Ishita, my true comrade-in-arms: thank you for being there through every grueling rebuttal and rejection; having someone who simply *got it* kept me sane. To Sat: our "radical idea" conversations pushed my perspective on life; thank you for making the journey both richer and more fun. To Neil, I am so glad we navigated our first conference together; thank you for always being down for a chat and for celebrating every win along the way. I am grateful for my ECE buddies, especially Turf, Jesse, and Dan, with whom I shared a vision for a more connected department. Starting the social committee together—and watch-

ing it grow into something that will outlast our time here—remains an enduring highlight. Separately, with Nick and many from that same crew, our NFL watch parties, Eagles games, and the chaotic fun of fantasy football became a much-needed escape. My thanks also go to those who shared different chapters of this journey: Lap, Bhishma, Rohan, and August, for the memories that defined my early years; Alex, Jan, and Stefan, whose depth, clarity, and dedication during the time our paths intersected left a lasting impression on me; and my roommates Sayeri and Margarita, for the late-night kitchen conversations that spanned everything from research to life and back again. You all made this journey what it was.

Nothing in my life stands apart from the love and care of my family. I don't know where to start with you; you have each shaped who I am today and mean immeasurably more to me than I can ever express. Joy, you have been my "second momma bear" from childhood onward. From reading this work in its infancy and sending me detailed notes to helping me with every single apartment move and cleaning my water filters, you have always gone impossibly above and beyond. You are the best older sister I could ask for. Paul, thank you for the example you set and for always believing in me. Your own journey has motivated me throughout my life, and I am grateful to share this one with you. Kara, I learn so much from your sharp mind and your clarity. Thank you for the way you look out for me, even as my little sister; your belief in my work and in me has meant the world. Lloyd, thank you for your reliability, your wisdom, and your endless thoughtfulness. I am grateful for the way you listen to and challenge my ideas; you constantly inspire me to be a better version of myself. To my four-pawed family—Melody, Muffin, Nabi, Willow, Maple, and Ginkgo—thank you for the simple joy and light you will always bring into my life.

Mom, you gave me *everything*. You are the strongest person I know, and I can never even begin to put into words what you mean to me. Your love and fierce support have been the foundation of my life. I only stand here because of you. Thank you, 엄마.

Parth, you have been by my side since the very beginning of this journey, from the deep friendship that budded over our long, early walks at Lakeside to the partner you are today: you walked with me every step of the way, including the moments only you know, with a depth of care I'll never take for granted. You lift me higher than I could ever go on my own. I am grateful to share this life with you.

To God alone be all glory and honor.

Bibliographical Notes

Most of the material in Chapter 2 originally appeared in a paper published in USENIX Security 2024[173], and was coauthored with Xiaoqi Chen and Jennifer Rexford. A smaller portion of Chapter 2 originally appeared as work published in SIGCOMM SPIN 2021[172], and was coauthored with Xiaoqi Chen. Chapter 3 is based on work originally published in NSDI 2024[27], coauthored with Henry Birge-Lee, Benjamin Herber, Jennifer Rexford, and Maria Apostolaki. Chapter 4 was developed in collaboration with Henry Birge-Lee, Sofia Marina, Jennifer Rexford, and Maria Apostolaki.

This work was funded by the National Science Foundation Graduate Research Fellowship Program (NSF GRFP), Princeton University's Wallace Memorial Fellowship in Engineering, the Defense Advanced Research Projects Agency (DARPA), and Protocol Labs.

1

Introduction

Today's Internet serves as a critical substrate for modern life, connecting billions of users worldwide to essential services, from communication and entertainment to education, healthcare, and other critical infrastructure. Despite this ubiquity, its underlying architecture and protocols are plagued by security vulnerabilities. Network services routinely suffer from resource attacks that disrupt or deny user access, while traffic analysis attacks compro-

mise user privacy. These issues endure as a direct consequence of fundamental limitations in the Internet's original design.

1.1 Internet Design and its Security Consequences

About half a century ago, the Internet was created as a research prototype connecting a small set of research networks whose operators held similar goals (*e.g.,* resource sharing and interoperable connectivity)—and were thus *incentive-aligned*. The network protocols introduced into this context were built on the premise that users and operators would act in good faith, and that no participant would deliberately exploit protocols for disruption or personal gain. In other words, *the Internet was not designed with security in mind*: it mainly considered threats in the form of physical disruption to infrastructure (*e.g.,* link or node failures), rather than attacks by network participants to exploit its underlying protocols.

With these assumptions as the backdrop, the Internet adopted several key design properties which, on the one hand, powered its massive success, but on the other, introduced the fundamental security challenges that remain deeply embedded today.

1.1.1 Plug-and-Play Design

The Internet's *plug-and-play design* makes it easy for any endpoint to send traffic: a host can transmit packets by simply specifying a destination IP address in the packet header, as shown in Figure 1.1. If the destination is publicly reachable, the network will forward the

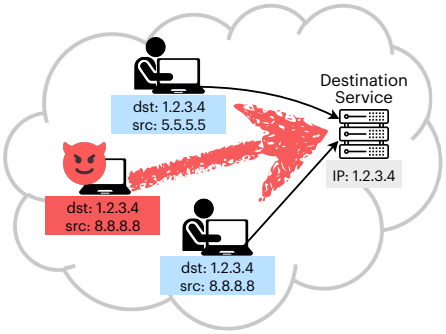


Figure 1.1: In the plug-and-play Internet, any host can address packets to any publicly reachable destination (blue). Consequently, the destination lacks mechanisms to restrict unwanted ingress traffic, leaving it exposed to attack (red).

packet based on the destination prefix, without any notion of sender authorization (*i.e.,* "who is *allowed* to send to that destination").

While open reachability has enabled broad access (*i.e.,* permitting *anyone* to connect) and rapid scaling (*i.e.,* making it *easy* for new hosts and networks to join), it comes with security consequences. Prominently, it enables *denial-of-service*—especially distributed denial-of-service (DDoS)–attacks. In such attacks, an adversary floods a victim with large volumes of malicious traffic to exhaust its finite resources (*e.g.,* bandwidth, CPU, or memory), preventing it from serving real clients (Figure 1.1). To make matters worse, the Internet does not provide a strong notion of sender identity: a packet's source IP address is not cryptographically authenticated, so it cannot be reliably bound to its true sender. As a result, attackers can forge source addresses (*i.e.,* source spoofing), making attribution difficult[1]. Effectively, the Internet provides no native mechanism to authenticate the

[1] In practice, a DDoS adversary may spoof the source addresses of all attack traffic to evade source-based filtering. Alternatively, it may distribute the attack across many devices to achieve the same effect.

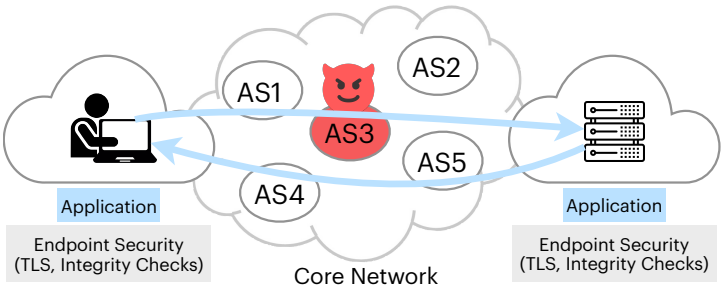


Figure 1.2: The end-to-end principle places functionality at the endpoints, enabling rapid innovation while keeping the core network simple. However, threats originating from within the core cannot be addressed by the endpoints.

origin or intent of a packet, beyond basic protocol semantics (*e.g.*, that it is formatted as a "request"). This leaves destinations inherently exposed to volumetric attacks, with no control over which traffic can reach them.

1.1.2 End-to-End Principle

The *end-to-end principle* is arguably the Internet's most foundational architectural principle. It holds that most "intelligence"—including security functionality—should reside at the endpoints (at the application and transport layers), while the network core remains as simple as possible and focused primarily on packet forwarding (see Figure 1.2, blue). A major benefit of this approach is evolvability. New applications can be deployed and existing ones can change through software updates at end hosts, without requiring upgrades across the thousands of independently operated *Autonomous Systems* (ASes) that constitute the network core. By decoupling innovation from infrastructure, this "dumb pipe" network model has been a major driver of the Internet's explosive growth.

However, this same design also has significant security consequences. Because the core was built as a simple forwarding fabric, it does not natively offer mechanisms to detect or constrain malicious behavior from adversaries *within* the network. As a result, on-path adversaries–located between communicating endpoints–can observe, delay, drop, replay, or selectively modify traffic with relative impunity (see Figure 1.2, red). While end-host protections like encryption (*e.g.,* TLS) can ensure payload confidentiality and integrity, they cannot prevent adversaries from exploiting properties of packet delivery and interdomain forwarding in the core that lie outside an endpoint's control. For example, even with encrypted payloads, network adversaries can exploit traffic *metadata*–such as patterns in packet timing, size, and directionality–to infer sensitive information about the underlying communication (*i.e., website fingerprinting attacks*), thereby compromising user privacy.

Addressing these threats remains a significant challenge, since endpoints have no direct control over routing and forwarding decisions in the network core. Existing endpoint workarounds, such as overlay networks, can only *indirectly* influence paths (*e.g.,* via tunneling or relaying). Additionally, these approaches incur heavy performance penalties because they introduce extra *software-based* per-packet processing: overlay endpoints must encapsulate and decapsulate packets (processing them in virtual layers), and relaying can detour packets through additional hops—collectively increasing CPU overhead and user latency.

1.1.3 Standardized Protocols

Standardized protocols are another key Internet design property. They enable global interoperability by defining common rules for how packets should be formatted, transmitted, and interpreted across different systems and independently operated networks. Because all participants speaks the same protocol "language" (*e.g.,* IP for addressing, BGP for forwarding, and TCP/UDP for transport), new networks and devices can join the global Internet and communicate seamlessly without additional coordination.

This reliance on widely adopted standards also carries important security consequences. Mainly, it makes protocol-level vulnerabilities hard to fix, since effective defenses require broad adoption to deliver meaningful protection. For example, routing-security extension BGPsec–proposed to address well-known weaknesses in the Internet's default routing protocol BGP–remains far from globally deployed years after standardization, limiting its practical benefit. Deployability barriers are steep: it is not possible to have an Internet-wide "flag day" for globally coordinated operational changes, and even incremental rollout depends on networks that may lack incentives to participate–or may not be trusted to do so.

In summary, fundamental network-level security problems persist today as direct consequences of the Internet's design. Plug-and-play connectivity enables ingress attacks from other endpoints, the end-to-end architecture offers limited protection against *in-network* threats, and standardized protocols impede mitigations by requiring global coordination.

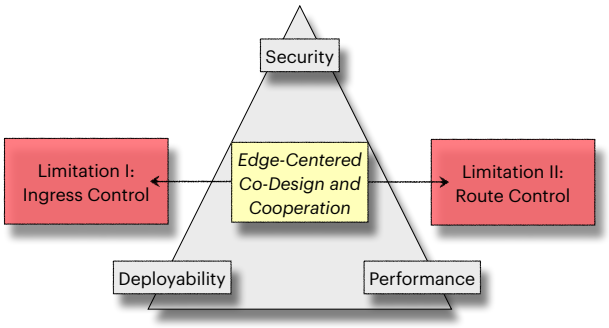


Figure 1.3: This dissertation's central idea is *edge-centered co-design and cooperation*: a practical way to address ingress- and route-control threats while satisfying critical security-deployability-performance constraints.

1.2 Protecting Users Within Internet Constraints

In this dissertation, we distill these security consequences into two fundamental limitations: limited *ingress control*, which prevents an end host from controlling *who* can send it traffic (leaving it vulnerable to availability attacks such as DoS/DDoS), and limited *route control*, which prevents an end host from controlling *how* its traffic traverses the core network (leaving it vulnerable to on-path privacy or performance attacks). Put differently, we target and address architectural vulnerabilities in how the existing network fabric admits and transports traffic—vulnerabilities that cannot be resolved by end-host mechanisms.

Addressing these limitations in practice is difficult because effective solutions must satisfy three competing constraints (see Figure 1.3): (i) improve *security*, (ii) remain *deployable* (*i.e.,* enabling incentive-compatible local adoption while preserving interoperability within standardized global protocols), and (iii) protect *performance* (*i.e.,* without degrading user experience via added latency or software overheads).

This dissertation resolves this tension through a unifying design approach—*edge-centered co-design and cooperation*—grounded in three underlying insights. (i) Moving defenses from the end host to the *network edge* (*i.e.,* the boundary between the local network and the wider Internet) provides practical leverage over traffic admission and interdomain routing. Crucially, however, this leverage alone falls short of control. (ii) To realize ingress control *within* a single network edge, *layered co-design* distributes complementary defense mechanisms across heterogeneous platforms to scalably enforce ingress policies at network speeds. (iii) To realize route control *between* multiple network edges, *selective cooperation* enables participating edges to expose additional paths beyond the default BGP route and jointly allocate traffic across them, without requiring global coordination. The remainder of this section develops these insights in turn and shows how, together, they form a secure, deployable systems framework for ingress control and route control on today's Internet.

1.2.1 From End Hosts to Edge Networks

The Promise of Edge Leverage

We begin by asking *where* defenses should be deployed to effectively address the outlined threats. While end hosts remain the natural place for many protections (*e.g.,* application-layer encryption), they have limited leverage: they are unable to prevent resource-exhaustion flooding attacks and they do not participate in interdomain routing. Addressing such

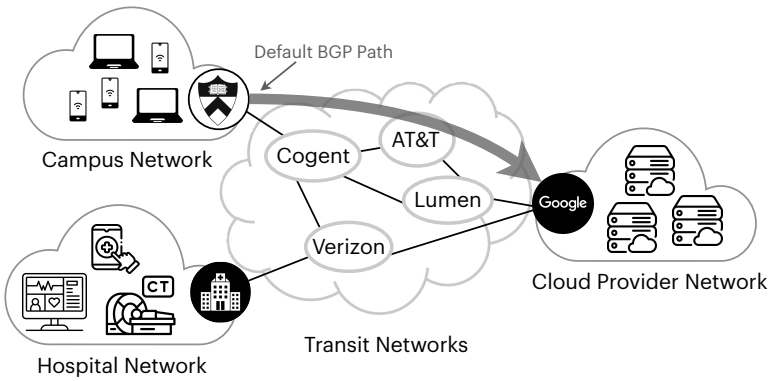


Figure 1.4: The Internet consists of independently administered domains across its edge and core. The *network edge* offers practical defense leverage: it sits at the domain's boundary and is managed by operators accountable to its users.

threats requires network-level leverage, making the *network edge* [2] the ideal starting point.

Edge placement lays the foundation for addressing ingress- and route-control security threats from within an administrative domain. In edge networks serving a particular user population (*e.g.,* a campus, hospital, or enterprise network), the network operator has both the authority and incentive to protect its users. Accordingly, the network edge—specifically, the border routers and switches where the edge domain connects to the wider Internet—is well-positioned to deploy defenses on behalf of downstream endpoints (see Figure 1.4). It can act on inbound traffic before it reaches end hosts or improperly consumes downstream resources. It is also the domain's point of presence for interdomain routing, where it advertises its own IP prefixes so that upstream networks can reach it and where it selects among available interdomain paths for outbound traffic.

[2] We distinguish the *edge network* as the full user-facing administrative domain, whereas the *network edge* is the boundary where that domain connects to the wider Internet.

The Limitations of Edge Leverage

In practice, edge networks today already deploy ingress security mechanisms at their borders, most commonly firewalls and intrusion detection/prevention systems (IDS/IPS). Firewalls regulate inbound traffic primarily by tracking connections initiated by *internal* hosts and permitting the associated return traffic. However, they are fundamentally ineffective as defenses for public-facing services such as web servers, which must accept connections initiated by *external* clients and remain exposed to volumetric availability attacks. IDS/IPS face a similar limitation: although they can filter ingress traffic based on pre-defined rules and anomalies, they are less effective when flooding traffic is well formed and protocol compliant. Filtering volumetric traffic for apparent "heavy hitters" is one possible mitigation, but it remains unreliable in practice, since attackers can evade such defenses through source spoofing, or by spreading traffic across large botnets so that no single source appears suspicious. As a result, existing edge mechanisms cannot reliably and scalably distinguish legitimate from malicious traffic, leaving public-facing services vulnerable to flooding attacks.

Prior literature has also explored pushing stronger volumetric defenses into edge hardware devices[105,177]. These works exploit the ability of network hardware to process traffic at line rate, enabling high-speed filtering before attack traffic reaches downstream services. However, defenses deployed at a single ingress point remain fundamentally limited. To

sustain line-rate processing, network devices—such as programmable hardware switches—must operate under tight constraints on memory and computational complexity. Existing hardware defenses struggle to operate within these constraints, oversimplifying logic and missing attack traffic [177] or introducing overheads that degrade user performance [105].

The limitations of single-point edge leverage extend beyond ingress control to route control as well. An edge's authority is inherently local to its administrative domain, and it does not control routing decisions at upstream networks. In the Internet core, routing is dictated by the Border Gateway Protocol (BGP), where transit networks determine paths based on their own economic incentives, rather than an edge's source preferences. These upstream networks export only their preferred route, and edges are offered only a single default BGP path. For example, as shown in Figure 1.4, there are, in principle, three transit paths available between the campus and cloud provider networks: (i) via Cogent, AT&T, and Lumen, (ii) via Cogent and Lumen, or (iii) via Cogent and Verizon. Yet, despite this apparent diversity, the campus edge cannot simply "pick" among these paths: it may only use the default BGP route via Cogent, AT&T, and Lumen.

Notwithstanding, it is operationally possible for an edge network to exert a highly constrained form of route selection through *multihoming*. With multihoming, an edge network connects to multiple upstream providers—typically two, and sometimes three in larger enterprise deployments—and can select among the routes those providers adver-

tise (*e.g.,* in Figure 1.4, the cloud network could select between paths offered by Verizon and Lumen). This enables local traffic engineering for the edge's first upstream hop. However, this control is fundamentally limited: the edge can only choose among its directly connected upstreams, restricting its influence to the initial segment of the interdomain route. It has little visibility into, and no control over, the remainder of the path that traverses deeper through the Internet core. Consequently, the edge cannot select alternative interdomain paths to steer traffic away from undesirable transit networks, even when they might pose risks to privacy or performance from on-path adversaries. And, of course, edge networks are often simply single-homed, leaving them without even this limited flexibility.

Taken together, these limitations illustrate that although the network edge is a promising point of leverage, a single edge platform cannot by itself provide either scalable ingress protection or substantive interdomain route control.

1.2.2 From Monolithic Defenses to Layered Co-Design

If the edge is indeed the right leverage point for ingress control—yet no single edge platform can provide scalable protection—how can we realize defenses that are at once secure, performant, and deployable? The answer is to *distribute* ingress enforcement across multiple platforms already operated by the edge and *coordinate* them to act as a unified defense. This grounds our second key insight: *layered co-design*, which partitions and carefully coordinates defense functionality across edge platforms with complementary capabilities.

Programmable hardware platforms (*e.g.,* PISA-based switch architectures such as Intel Tofino[81]) can execute simple packet-processing primitives at extremely high throughput, making them well-suited to absorb massive volumes of adversarial traffic. However, as discussed previously, network hardware operates under strict constraints, such as limited on-chip memory and restricted instruction sets, which make complex per-packet logic—especially related to security—difficult to implement. Software platforms offer the opposite tradeoff: they support richer policy logic and sophisticated stateful processing, but cannot sustain adversarial packet rates without severely degrading performance.

Layered co-design exploits the strengths of both to achieve major scaling wins: early hardware layers block volumetric attack traffic with high-speed, coarse-grained filtering, while downstream software layers process the small volume of remaining traffic with slower, fine-grained verification, preserving overall exactness without sacrificing throughput.

While layering the defense across heterogeneous platforms gains scalability, it also introduces a new systems challenge: the defense is physically distributed, but it must behave as a unified system. Each platform has distinct resource constraints and processing capabilities, yet together they must coordinate to securely distinguish between wanted versus unwanted traffic, efficiently enforce filtering decisions, and consistently maintain enforcement state—all while operating at line rate. Further, each layer must remain secure despite the hardware layer's limited support for implementing security primitives: attackers must not be able to

exploit weaknesses in either layer to bypass filtering and evade enforcement.

To address these challenges, we co-design a lightweight coordination protocol between both platforms. This protocol allows our distributed components to exchange compact enforcement signals and maintain consistent state without introducing delays or processing overhead. Because this coordination occurs entirely among platforms jointly administered by the edge network, the implementation remains localized, requiring no modifications to upstream infrastructure or downstream end-hosts. Finally, to secure the operations of the hardware layer within its strict resource constraints, we introduce a lightweight switch-based cryptographic primitive that serves as our building block for secure in-network traffic classification. In short, by partitioning functionality in this manner, layered co-design enables a unified ingress enforcement architecture that scales to massive attack volumes, without sacrificing flexibility or interoperability.

1.2.3 From Global Coordination to Selective Cooperation

To address ingress-control threats, we distribute enforcement across platforms within a single edge domain, leveraging support from *downstream* components. Route-control threats, on the other hand, call for support from *upstream* networks: although an edge can announce its own prefixes, upstream transit networks ultimately determine the interdomain paths under BGP. Importantly, different interdomain paths can potentially expose traffic to distinct privacy and performance risks, which the edge cannot meaningfully address on

its own. This raises a central question: how can we address route-control threats—an inherently interdomain problem—in a way that remains secure, preserves performance, and is still deployable without Internet-wide coordination?

Our answer—and final key insight—is *selective cooperation* among *incentive-aligned* parties. Although today's Internet is no longer globally incentive-aligned—having grown from a small, reachability-driven network into one comprised of tens of thousands of independently operated networks with distinct, often conflicting, policies and economic interests–smaller coalitions of networks may still locally share incentives. For example, such networks may share operational objectives, serve overlapping user populations, or bear similar reputational and financial costs of abuse. Selective cooperation therefore enables coordination where incentives already exist, without requiring global trust or universal participation.

At a high level, selective cooperation allows cooperating edges to move beyond the single default BGP path and expose alternative interdomain paths. A receiving edge can advertise *multiple* prefixes per destination and bind each prefix to a distinct interdomain path using BGP-compliant announcements that constrain how the routes propagate through the network. A cooperating sending edge can then select among these paths by simply specifying the relevant destination address in outgoing packets. Intermediate networks continue to run BGP unchanged, routing packets based on destination prefix as usual.

Exposing alternative paths, however, is only the first step. To make effective use of them,

cooperating edges must assess the newly available paths in terms of privacy risk and performance cost, and then determine which paths to use. This is challenging because, although the edges can coordinate with one another, the transit networks between them remain untrusted and non-cooperating, and may distort or manipulate the signals needed to measure path quality reliably. Moreover, the right traffic allocation is not obvious: edges must jointly reason about the risk associated with each path or combination of paths, how much traffic to place on each, and how to balance privacy risk against performance cost.

To address these challenges, we design a lightweight coordination protocol that allows cooperating edges to exchange reliable control information and operate jointly despite the untrusted networks between them. The protocol enables secure end-to-end measurements using lightweight in-network cryptographic mechanisms, allowing edges to detect tampering and obtain reliable path-performance data. Building on these measurements, we develop a modeling and optimization framework that determines how traffic should be allocated across the available paths in order to balance privacy and performance objectives. Importantly, this approach delivers immediate benefits under partial deployment: cooperating edges gain improved path control, while non-participating networks require no protocol changes. Together, these mechanisms enable edges to exercise meaningful control over interdomain path selection, without relying on Internet-wide coordination.

1.3 Dissertation Roadmap

This dissertation develops an edge-based framework for protecting users against threats rooted in two persistent Internet limitations: limited *ingress control* (over which traffic can be sent to an endpoint) and limited *route control* (over how traffic traverses the Internet core). Our approach prioritizes deployable mechanisms that respect operational constraints —including limited interdomain trust and partial adoption—while also protecting user performance. Accordingly, we organize this dissertation into two complementary parts: (I) addresses ingress-control threats via *intra*-edge co-design, and (II) addresses route-control threats via *inter*-edge cooperation. Taken together, these systems provide a practical foundation for improving ingress and route control under real-world constraints.

In Part I, we show how ingress control can be enforced at the edge by combining fast, hardware-accelerated early defense layers with slower, software-based processing in later defense layers. The central challenge we overcome is securely identifying unwanted (*i.e.,* attack) traffic at scale and acting with sufficient speed, all while protecting benign user traffic and remaining within the compute and memory limits of modern networking platforms.

- In Chapter 2, we instantiate our approach with SmartCookie, a tiered defense architecture that combines coarse-grained *approximate* filtering in hardware with fine-grained *exact* processing in software, to mitigate an important class of volumet-

ric attacks called SYN flooding. We also present a switch-native hashing primitive (§2.5) that serves as a cryptographic building block for SmartCookie to perform secure in-network traffic classification, distinguishing traffic as malicious or benign.

Part II of this dissertation tackles route control objectives that span administrative domains. Because a single domain cannot unilaterally dictate interdomain paths, we leverage cooperation among pairs of incentive-aligned edge networks to expose and optimally select from a diverse set of routing options using lightweight, edge-to-edge mechanisms.

- In Chapter 3, we introduce Tango, which enables cooperating domains to discover and expose additional interdomain routing options via selectively propagated, BGP-compliant advertisements. Tango binds each routing option to a distinct destination prefix (multiple prefixes per destination), so that different prefixes induce different interdomain paths. Once exposed, cooperating edges can steer traffic among these options with edge-to-edge tunneling: using fast, hardware-based encapsulation/decapsulation, they can simply select alternate paths by specifying a tunneled destination address corresponding to their desired path.

- Finally, in Chapter 4, we present PraxiGuard, which systematically models the privacy risk and performance cost of exposed routing options, and determines how traffic should be split across them to achieve operator-defined privacy and performance objectives. PraxiGuard thus identifies how newly available paths should be

used to optimally balance privacy and performance, and evaluates both the opportunity and the practical benefit of traffic splitting for mitigating on-path threats.

Together, these chapters show how edge-based mechanisms can strengthen user protection under real deployment constraints: Part I develops scalable intra-edge enforcement for controlling which traffic is allowed to reach endpoints, while Part II develops lightweight inter-edge mechanisms for exposing, selecting, and allocating traffic across interdomain paths. In this way, the dissertation advances a unified framework for improving both ingress control and route control on today's Internet.

Part I

Ingress Control via

Intra-Edge Co-Design

2

Blocking Large-Scale Flooding Attacks with Tiered Line-Rate Filtering

In this chapter, we present SmartCookie, which mitigates an important class of volumetric attacks called *SYN* flooding. SmartCookie instantiates a novel tiered defense strategy, leveraging emerging programmable switches to block 100% of attack traffic in the switch

data plane, along with state-of-the-art kernel technologies such as eBPF, which together enable *scalability* for serving benign traffic. We also introduce a switch-native hashing primitive, that—to the best of our knowledge—makes SmartCookie the first system to run cryptographically secure SYN cookie checks on high-speed programmable switches, protecting both *security* and *performance*. SmartCookie defends against adaptive adversaries at two orders of magnitude greater attack traffic than traditional CPU-based software defenses, blocking attacks of 136.9 Mpps *without packet loss*. We also achieve 2x-6.5x lower end-to-end latency for benign traffic compared to existing hardware-only defenses.

2.1 Introduction

Distributed Denial-of-Service (DDoS) attacks have been studied for decades, but despite this rich history, volumetric attacks are still an important and unsolved problem today. In 2023, attacks increased by over 300%[84,1], with downtime costing companies an average of $20,000-$40,000 hourly[65,52]. One driving factor behind this growth is the widespread availability of DDoS-for-hire services, costing as little as $10 hourly and making it increasingly easy to launch attacks[70]. In recent years, DDoS attacks have also shaped political landscapes and played major roles in cyber warfare[59,51,49,76].

One of the most common DDoS attacks, namely *SYN floods*, consume server memory until the server is forced to drop benign traffic[73,74,123]. SYN floods constituted up to 94.7%

of all DDoS attacks in 2020[96] and continue to remain a critical threat today[97]. Additionally, benign traffic volumes also continue to grow exponentially, reaching staggering loads of up to hundreds of Gbps in cloud-provider networks[134].

To respond to growing threats and traffic volumes, network providers (*e.g.,* cloud providers, enterprise networks, ISPs) urgently require *scalable* defenses that block volumetric attacks *without compromising security* against adaptive adversaries or *degrading application performance*[80,123,47]. In other words, they need defenses with three key requirements: **security** (blocking attacks from adaptive adversaries), **scalability** (handling large amounts of benign and attack traffic), and **performance** (low latency for benign clients).

While state-of-the-art SYN-flooding defenses using *SYN cookies* have been proposed and standardized for many years, existing solutions have failed to simultaneously provide security, scalability, *and* performance[104,154,157,105,177]. Designing and practically implementing modern SYN-flooding defenses that meet each of these requirements is challenging. Compared to defenses against many other volumetric attacks (e.g., ACK floods, RST floods, UDP amplification), SYN-flooding defenses cannot simply drop, rate-limit, or ignore unsolicited traffic. Doing so might cause denial of benign client requests, particularly since adversaries often spoof attack packets using legitimate addresses. Instead, defenses must identify and block large-scale attacks (requiring costly compute) while keeping per-flow state to track large numbers of verified connections (requiring large chunks of memory).

Server-Based Solutions. Server-based SYN cookie defenses are the time-honored way to provide security and can scalably handle *benign traffic* by default [104,154,157]. However, under large-scale attacks, software-based packet-processing and cryptographic cookie computation incur high overheads and lead to CPU exhaustion (§2.8). Ultimately, once the server's limited CPU capacity is overwhelmed by *volumetric attacks*, these defenses collapse and again result in DoS.

Switch-Based Solutions. High-speed programmable hardware switches present a unique opportunity to overcome this additional attack vector on end host CPUs. Switches provide packet-processing at orders of magnitude faster rates, and thus have the potential to execute the required cryptography more performantly than end hosts. However, purely switch-based solutions [105,177] must operate under strict hardware constraints, including limited compute and limited memory. First, cryptographic primitives are not natively supported in switches and are computationally expensive. As a result, existing switch-based defenses use the insecure CRC32 to compute SYN cookies [105,177], *abandoning security* and defeating the purpose of the SYN cookie check (§2.2.4). Additionally, because switches have only tens of MBs of memory [94], memory-intensive SYN-flooding defenses which track per-connection state *struggle to scale* [177] or *degrade application performance* [105] (§2.2.5).

Contribution

To overcome the limitations of existing solutions, we propose **SmartCookie**: a novel *tiered* defense that leverages collaborative programmable data planes on hardware and software targets. Our design *intelligently* partitions the defense workload to maximize the benefits of both switch- and server-based defenses, while minimizing the limitations of each.

Our key insight for motivating SmartCookie's novel division of labor is two-fold. Switches are highly performant and have potential to quickly and *securely* block volumetric attacks, but they are memory-limited and scale poorly at keeping exact state for verified flows. This makes switches excellent as a first line of defense, but they should not be required to keep per-flow state. Meanwhile, servers enjoy superior memory resources, but are prohibitively slow at packet processing. This makes them ideal for exactly tracking *benign flows*, without the burden of blocking *attacks*.

At a high level, SmartCookie takes traditional SYN cookie defense elements, refactors them, splits them between a co-designed *switch agent* (running on the switch data plane) and *server agent* (running on the Linux kernel data plane), and stitches switch agent and server agent together with a collaborative protocol. We identify three key elements of existing defenses: *F1*) *SYN cookie checks*, *F2*) *TCP sequence number translations*, and *F3*) *keeping state for verified connections*. Following our key insight, we refactor and map defense elements *F1-F3* to switch and server as follows: SmartCookie switch agent *securely* performs

cookie checks to quickly stop bad traffic (*F1*) and *approximately* tracks verified connections (*F3.A*) instead of keeping exact state, while SmartCookie server agent handles sequence number translations (*F2*) and *exactly* tracks verified connections (*F3.B*).

SmartCookie switch agent maintains approximate state using compact data structures with well-defined accuracy guarantees (§2.6.3). Note that in contrast to prior work, our defense does not require exact state at the switch, but *still achieves overall exact defense results*, due to its split design.

Summary and Roadmap

We summarize the chapter as follows:

- Section 2.5 introduces an in-network cryptographic hashing primitive to run cryptographically secure SYN cookies on programmable switches, instead of the insecure CRC32 used in prior works.
- Section 2.6 presents the first split-proxy SYN-flooding defense for modern programmable data planes, using in-switch compact data structures for memory scalability and server-side eBPF for immediate deployability.
- Section 2.8.1 implements an end-to-end SmartCookie prototype with a *switch agent* in P4 on Tofino switches and a *server agent* in eBPF on Linux servers.

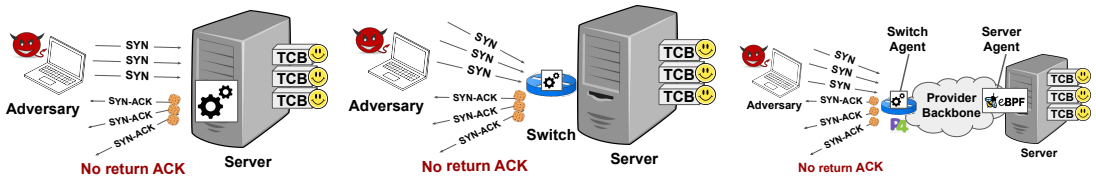
- The remainder of Section 2.8 evaluates this prototype, showing resilience against attack rates of 136.9 Mpps (about 92 Gbps) *without any packet loss*, with potential for easily reaching even higher rates. We also show 19×–105× throughput improvement over *software* solutions resulting from offloading of cookie computation and verification to high-speed switch hardware (§2.8.2), and 48-84% latency improvement with our novel split design over existing *hardware-only* designs (§2.8.3).

We present limitations of existing defenses in Section 2.2 and describe the threat model, problem setting, and architecture of SmartCookie in Section 2.3 and Section 2.4. We include a security analysis in Section 2.7, discuss additional features and future directions in Section 2.9, present related work in Section 2.10, and conclude in Section 2.11.

2.2 Limitations of Existing Defenses

2.2.1 Limitations of Server-Based Defenses

Traditional server-based SYN-flooding defenses use *SYN cookies* to protect server memory, encoding state normally saved to server-side Transmission Control Blocks (TCB) during connection setup in a cryptographically computed cookie instead[104] (Figure 2.1a). The server uses the SYN cookie as the initial sequence number (ISN) that is placed in the sequence number field (`seq_no`) of the SYN-ACK packet sent to the client. Clients automatically return the cookie in the acknowledgment number of the final ACK of the TCP



(a) Server-based defense. **(b)** Switch-based defense. **(c)** SmartCookie: a split defense.

Figure 2.1: SmartCookie versus server-based and switch-based defenses.

handshake, as `ack_no` = `seq_no` + 1. This returned cookie is then verified by the server.

In server-based defenses, the server cryptographically computes and verifies cookies and performs packet processing for every potential connection, introducing an added attack vector on server compute resources. Because these defenses run on general-purpose CPUs, they can easily be overwhelmed under heavy attack loads, due not only to cookie computation but also to software-based packet-processing overheads (§2.8, Table 2.1). Thus, server-based defenses struggle to scalably stop volumetric attacks, degrading application performance and again resulting in denial-of-service for benign clients.

2.2.2 Limitations of SYN Cookie Proxies

To protect the server from CPU exhaustion, the SYN cookie mechanism can run in a proxy placed in the NIC of the server or in a switch near the server (Figure 2.1b). Proxies perform cookie computation and verification in place of the server, only establishing a connection with the server for verified clients and thus defending against attacks that cause excessive consumption of server CPU cycles[69,130]. Defense proxies also monitor previously verified

connections to determine which packets require cookie checks.

However, this approach has fundamental drawbacks as well. First, proxies incur overheads related to connection setup. Specifically, after a proxy verifies a client, it must perform a second, separate proxy-to-server setup handshake to establish the connection from the server's point-of-view. This increases latency since the proxy must buffer any packets received from the client while it establishes the server-side connection.

Additionally, proxies need to translate TCP sequence numbers between client and server, because there is a mismatch between the ISN chosen by the proxy as the SYN cookie value and the ISN chosen randomly by the server[130]. This forces the proxy to keep exact per-flow state to track verified connections (in addition to the per-flow state kept by the server), and the proxy must perform costly sequence number translations on every packet throughout the remainder of the connection. We note the SYN proxy design also relies on assumptions of symmetric routing, since all packets must pass through the proxy in both directions for sequence number translations.

2.2.3 Opportunities of Programmable Switches

Programmable switches are similar in price to fixed-function enterprise-grade switches. They are an attractive option for high-speed, flexible, and cost-efficient defenses against large-scale attacks[44,43,181,177,105]. While it is clear that switches have significantly more potential than servers for a performant and robust defense, we argue that programmable

switches also showcase concrete benefits over SmartNICs as well.

Target Choice. Although some state-of-the-art SoC-based SmartNICs offer larger memory banks and cryptographic accelerators for specific primitives (e.g., MACsec/IPsec encryption) [114], switches have faster packet processing and offer better performance to cost/power ratios than alternative SmartNIC approaches [105,114,90]. For our target setting of large cloud enterprises, gateway switches can save bandwidth in the network core by blocking attacks directly at the network edge, before malicious traffic reaches the internal network and hosts' NICs. Still, SmartCookie could be deployed on SmartNICS, but given their lower capacity and better fit for smaller edge networks, we target switches.

2.2.4 Security Limitations of Switch Defenses

Prior works have leveraged high-speed switches as SYN cookie proxies [105,177]. Unfortunately, these systems have a crippling security flaw: they use the insecure CRC32 checksum as the "hash" for computing SYN cookies, abandoning security against adaptive adversaries.

The Need for a Secure Hash. SYN cookies are computed with a hash of the connection 4-tuple (source/destination IP addresses and port numbers), along with some secret key and a timestamp, so an adversary cannot perform a replay attack with the cookie at a later time. For security, this hash must be cryptographically robust, such that an adversary cannot easily craft a cookie that would pass as legitimate during a cookie check. Otherwise,

the adversary could send many forged cookies to the victim, launching an attack more powerful than the original one: after verifying forged cookies, the victim must also prepare for new connections and allocate memory, wasting *both* compute and memory resources.

Insecurity of CRC. CRC is an error-detection checksum, not a cryptographic hash. When incorrectly used as such, it can be trivially cracked by an adaptive adversary in key recovery and cookie forging attacks[166]. There are two broad attack vectors against CRC that can result in breaking the cookie defense: collision induction and nonce deduction.

Collision Induction. An adaptive adversary can exploit the fact that CRC is not collision-resistant to construct hash collisions. Without assuming security by obscurity, adversaries with knowledge of a CRC-based defense can simply send probe packets with different inputs to uncover a hash collision. Note the adversary has constant feedback on CRC outputs, as the defense must respond to all SYN packets.

Nonce Deduction. Due to its linear nature, CRC is vulnerable to nonce deduction; an adaptive adversary can trivially crack CRC-based cookies in a few simple steps, as we detail next. Given the connection 4-tuple m and the secret nonce n, the CRC-based cookie is typically generated using $\mathrm{CRC}(m||n)$. We note CRC is linear and affine: for any A, B,

and C we have

$$\mathrm{CRC}(A) \oplus \mathrm{CRC}(B) \oplus \mathrm{CRC}(C) = \mathrm{CRC}(A \oplus B \oplus C). \tag{2.1}$$

The adversary can send a SYN packet with known input m_0 and observe the output $\mathrm{CRC}(m_0 \| n)$. Subsequently, the adversary can compute $\mathrm{CRC}(m_0 \| 0)$ herself, and derive

$$\mathrm{CRC}(0 \| n) = \mathrm{CRC}(m_0 \| 0) \oplus \mathrm{CRC}(m_0 \| n) \oplus \mathrm{CRC}(0). \tag{2.2}$$

Now that the adversary knows $\mathrm{CRC}(0 \| n)$, she can compute new cookie value for any arbitrary 4-tuple m_x, as follows:

$$\mathrm{CRC}(m_x \| n) = \mathrm{CRC}(m_x \| 0) \oplus \mathrm{CRC}(0 \| n) \oplus \mathrm{CRC}(0). \tag{2.3}$$

We also note that the same attack can be applied even if the nonce is placed at different input locations. The simplicity and effectiveness of this attack leaves the door wide open for adaptive adversaries to completely break the CRC-based SYN cookie defenses of both [105] and [177]. In contrast, SmartCookie *securely* computes SYN cookies in the programmable data plane using a robust hash with strong security guarantees [19] (§2.5).

	Secure Hash	High-Throughput	Non-Disruptive	Scales Beyond Switch Memory
Server-Based Cookies [104,130,120]	✓	✗	✓	–
Poseidon [177]	✗	✓	✓	✗
Jaqen [105]	✗	✓	✗	✓
SmartCookie	✓	✓	✓	✓

Table 2.1: Compared to existing defenses, SmartCookie is secure, high-throughput, non-disruptive, and scalable.

2.2.5 Performance Limitations of Switch Defenses

Existing switch-based defenses also struggle to provide memory scalability and good application performance [105,177].

Poor Scalability. Poseidon [177] keeps per-flow state in the switch to perform sequence number translations for ongoing flows. Given limited switch memory, this simply cannot scale, as processing hundreds of thousands of flows at network speeds is a memory-intensive task (Table 2.1). In contrast, SmartCookie avoids keeping exact state in the switch by offloading it to the server, allowing the defense to scale (§2.6).

Disruptive Performance. Jaqen [105] also avoids keeping per-flow state on switches. However, it does so by disruptively forcing all benign clients to undergo a reset during connection setup, even after passing the SYN cookie check (Table 2.1). After this reset, clients eventually attempt a second handshake, which the Jaqen proxy allows through to

the server, with some probability of false positive error. This enables server and client to directly choose sequence numbers without involving the Jaqen proxy. Unfortunately, Jaqen's design incurs extra latency (1-2 round-trip times) for benign connections, an especially undesirable performance penalty for clients across wide-area networks. In contrast, SmartCookie is transparent to clients, requiring a *single* handshake and maintaining good application-level performance (§2.8).

2.3 SmartCookie Problem Setting

The problems that state-of-the-art defenses experience against large-scale SYN floods are challenging to resolve, because they arise from the inherent hardware constraints of available targets and from the threat model experienced by network providers tasked with providing security, scalability, and performance. SmartCookie overcomes these challenges and presents a practical defense for large network operators, who control backbone switches and servers in their network. This setting opens unique opportunities for a division of labor that SmartCookie intelligently exploits (Figure 2.1c).

2.3.1 Threat Model

There are four key players in our threat model: clients, adversaries, switches, and servers. Switches and servers are controlled by the same network operator, which seeks to protect its servers from resource strain, while concurrently protecting network bandwidth. Under the

same administrative authority, switches and servers can safely cooperate.

Clients. Client communications to any of the servers hosted by the provider backbone should be protected from SYN flooding disruption. Clients should also not experience degraded performance as a result of any deployed defense.

Adversaries. Our threat model focuses on *asymmetric* SYN-flooding attacks, where adversaries require orders of magnitude fewer resources than defenses. Asymmetric attacks are generally regarded as more challenging to defend against[39,55], and their lower cost is likely what makes them so prevalent in the wild[97]. In our threat model, adversaries can spoof source IP addresses or utilize a limited number of compromised devices to send a flood of connection handshake requests that appear to be from unique clients and must be individually handled[9]. When spoofing, adversaries cannot gain feedback from the defense, as response packets to spoofed addresses never reach the adversary. Adversaries can also send some probe packets without spoofing to observe the resulting cookies and attempt to crack the cookie hash, as well as launch replay attacks using earlier cookies (§2.2.4). We assume the adversary can send attack traffic at up to Tbps rates. The adversary does not have physical access to the network switches or servers hosted by the provider backbone, and traffic from the adversary cannot reach the server without traversing a participating switch. In other words, the adversary cannot tamper with packets between the switches and

servers internal to the provider backbone.

Switches. The switches are programmable, high-speed network hardware capable of Tbps processing rates. They are controlled by the same network provider and deployed at the network edge. Since the switches and servers are under the same *centralized* operator, we assume the communication channel between the switches and the servers is secure.

Servers. Physical servers, hereafter simply called servers, are owned, operated, and trusted by the network provider, and thus can be modified. Trusted defense modules run on the servers, but for immediate deployability, changes to the TCP/IP stack of the server's Linux kernel are not required. However, modest changes to the server TCP/IP stack can open further defense opportunities in the future (see §2.9). We note that tenant VMs running on the physical servers are not trusted and thus are *unmodified* by the network provider. Also, in order to not affect the performance of applications running on the server, it is critical that any defense mechanism does not consume excessive CPU cycles.

2.3.2 Challenges

Modern programmable switches support flexible packet processing optimized for Tbps speeds[81,33,87]. They exert fine-grained control over packet forwarding in the data plane with line-rate throughput guarantees, but do so at the cost of strict constraints enforced by the

underlying hardware[81,32,31]. Thus, there are fundamental challenges to overcome to realize the potential of a high-speed switch defense.

Challenge 1: Limited In-Switch Programming Model for Cryptographic Operations. Most programmable switches do not natively support any cryptographic primitives, and they can only process packets with a limited number of available operations (e.g., addition, subtraction, and XOR, but no multiplication or division). However, even more fundamentally, to guarantee high-throughput, switches use a pipeline model with only a limited number of stages for packet processing. Operations can also only be performed concurrently in a stage if there are no dependencies between operations. Finally, the output of one computation cannot be used until the following pipeline stage, making it difficult to fit all the necessary operations for cryptographic SYN cookie computation within the limited number of available stages. Thus, even if cryptographic accelerators were introduced on hardware switches, computing secure cookies within the computational constraints and limited stages is challenging, and could break the performance of the switch if done naively. We show how SmartCookie overcomes these challenges in §2.5.

Challenge 2: Limited In-Switch Memory for Per-Connection State. The amount of available data plane memory is limited, and it must be shared with other applications (e.g., routing tables) running on the switch. To perform sequence number trans-

lations and maintain the correctness of packet-processing, a naive defense would need to keep cumbersome per-flow state on the order of 6 bytes per connection (e.g., the 32-bit sequence number and hashed connection 5-tuple key). Unlike other data plane applications that require keeping significantly less state (e.g., a small 6-bit version number along with a hashed 5-tuple key) [108], the amount of state that must be kept by a defense cannot be further compressed with a hash digest, as this would cause information loss and break connections. Given available memory is on the order of tens of MBs, keeping this amount of state for every verified flow in a large network provider cannot scale to even hundreds of thousands of connections [105], and thus we simply cannot afford to allocate switch memory for each ongoing TCP flow. We show in §2.6 how SmartCookie intelligently partitions the defense to gracefully handle this challenge.

2.4 SmartCookie Architecture

To overcome the challenges and existing limitations, SmartCookie proposes a novel split-proxy architecture (Figure 2.2).

Switch Agent. The SmartCookie switch agent performs cookie checks *securely* (F1) and tracks verified connections *approximately* (F3.A). This design is motivated by the underlying switch architecture, which is optimized for high-speed packet processing up to Tbps, an order of magnitude greater than the speeds of general-purpose CPUs. The switch

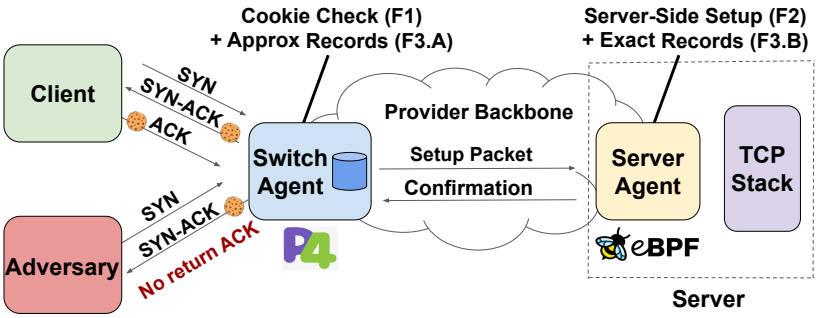


Figure 2.2: SmartCookie's split-proxy architecture.

agent is an ideal location for offloading of SYN cookie checks. It can securely and performantly block *100% of SYN floods*, without burdening the server (§2.5,§2.8). Recall that to achieve high throughput and low latency, however, the switch only has a limited amount of memory, and memory accesses are constrained. Thus, it should not be required to remember all the connections that have successfully passed the cookie check. In other words, *verified records* should be kept in an *approximate* data structure at the switch agent (§2.6.3).

Server Agent. Meanwhile, SmartCookie server agent is primarily responsible for handling benign connections, conducting sequence number translations (F2) and *exactly* tracking verified connections (F3.B). Servers offer greater compute flexibility and have fewer constraints on memory access and usage than switches. Thus, the server agent is ideal for maintaining exact information about verified connections and for performing sequence number translations on behalf of the switch agent. This offloading of unique per-connection sequence number translations is enabled by a special setup procedure, which

the switch agent and server agent cooperatively perform using custom setup request and confirmation packets (§2.6). The server agent's sequence number translation mechanism ensures consistent sequence number progression, preserves the correctness of the TCP protocol, and is transparent to unmodified end hosts. We note that the greater compute flexibility at the server agent comes at the cost of lower packet-processing speeds as compared to hardware speeds, but this is a reasonable tradeoff since the amount of benign traffic the server agent must process is much smaller than the amount of attack traffic that the switch agent must identify and drop.

2.5 Secure SYN Cookies in the Data Plane

SmartCookie switch agent is responsible for performantly computing and verifying SYN cookies in the data plane, but for any defense to be worthwhile, this must be done securely. To this end, we implement a secure keyed hash function, HalfSipHash, optimized for commodity programmable switches. Our approach exploits dependency management schemes to conserve pipeline stages and slicing-based bit manipulations for concise circular shifts.

2.5.1 Choice of Hash Function

Recall that because of the computational constraints of the switch, it is extremely challenging to compute a cryptographically secure hash function in the data plane. Hence, several prior works [105,177] opted to use the CRC32 checksum as a "hash" function to compute

SYN cookies, resulting in significant vulnerability [166]. This is because an adaptive adversary can always send crafted SYN packets and observe the resulting cookies (i.e., a chosen-plaintext attack), efficiently solve and extract the key used in hashing, and then forge cookies for any 4-tuple, bypassing the defense entirely (§2.2.4).

SmartCookie instead computes and verifies cookies using *HalfSipHash-2-4* [19], which is from the SipHash family of hashes used by Linux for computing SYN cookies [104]. SipHash is a keyed pseudorandom function designed for high speed on short inputs [19], making it well suited to packet-header fields such as flow tuples. SipHash-*c*-*d* initializes four internal 64-bit state variables from an input string and a 128-bit secret key, performs c compression rounds on the input, and then applies d finalization rounds to produce the output.

$v0 = k0 \oplus 0x70736575$
$v1 = k1 \oplus 0x6e646f6d$
$v2 = k0 \oplus 0x6e657261$
$v3 = k1 \oplus 0x79746573$

Table 2.2: HalfSipHash initialization.

HalfSipHash is the 32-bit variant of SipHash. It uses the same overall construction, but operates on 32-bit words with a 64-bit key and different shifting constants. Four internal 32-bit words, $v0$, $v1$, $v2$, and $v3$, are initialized by XORing the upper and lower 32-bits of the 64-bit key with four 32-bit fixed constants, as shown in Table 2.2. Each 32-bit input word is then mixed into the state and processed through c SipRounds, where each SipRound updates the internal state using additions, XORs, and circular left shifts, as

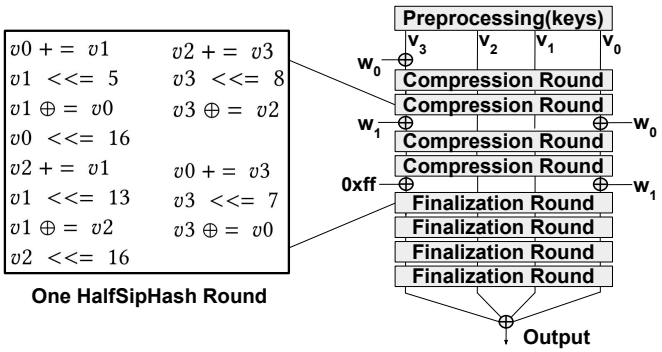


Figure 2.3: HalfSipHash-2-4 requires 2 compression rounds on each input word w and 4 finalization rounds at the end, with each round costing 14 operations.

shown in Figure 2.3. After all input words are processed, the algorithm performs d finalization rounds, and finally the 32-bit output $[v0 \oplus v1 \oplus v2 \oplus v3]$ is returned.

HalfSipHash-c-d is expected to provide maximum security for any keyed hash function with the same key size and output size, given compression and finalization rounds with $c \geq 2$ and $d \geq 4$, respectively[18]. By design, HalfSipHash performs most optimally for short input messages. We note that many programmable switch applications are interested in hashing packet header fields (not full packet payloads), and thus would benefit greatly from HalfSipHash's excellent security and speed on short inputs. Additionally, HalfSipHash in the data plane would be good for applications that might only need to hash a small fraction of packets, such as when a connection is first established. We also note that first-generation P4 programmable switches store variables in 32-bit containers natively. Since HalfSipHash operates on 32-bit words, this makes it an attractive choice for data-plane applications that require *security* and *high performance*.

In summary, we choose HalfSipHash-2-4 for performance reasons, as it is faster than SipHash-2-4, while still sharing the same construction. We believe HalfSipHash-2-4 *with key rotation* offers acceptable security against even well-provisioned adversaries (e.g., key brute-forcing adversaries), achieving good security along with lightweight performance. The hash is cryptographically robust and designed for speed on short inputs, making it ideal for computing cookies in the data plane, where the input to the hash is just a few bytes from the packet header.

2.5.2 Implementing a Cryptographic Hash

We now present our implementation of HalfSipHash on P4 programmable switches. We focus on HalfSipHash-2-4 due to its good security-versus-speed tradeoff, but our design can easily be used to support other HalfSipHash-*c*-*d* variations. Our design overcomes two challenges: limited number of switch pipeline stages, and limited arithmetic operations.

Limited Number of Pipeline Stages. Programmable switches have a limited number of pipeline stages available. Within each pipeline stage, the program can perform several arithmetic operations (such as add, xor, shift, etc.) concurrently on different variables. However, the output results are not available for use until the next pipeline stage, so other operations depending on these results must wait to be processed in this next stage.

HalfSipHash iteratively updates four internal state variables in repeated SipRounds,

which creates a long, interwoven dependency chain. Naively implementing SipRound per the algorithm in Figure 2.3 would thus cost too many stages or even consume the entire pipeline. To fit a SipRound within the switch processing pipeline, we help the compiler recognize dependencies by making them explicit with variable renaming and by grouping operations based on their dependencies. We also use packet recirculation and combined ingress + egress pipeline stages to further optimize HalfSipHash for the data plane.

Variable Renaming: We use a separate set of variables $a0$ - $a3$ and $b0$ - $b3$ in addition to the internal state variables $v0$ - $v3$ so that all operations will have distinct input and output variable names. This makes the dependency relationships between the different operations more clear, and also makes it easier for us to arrange the operations into separate actions for the P4 compiler.

Dependency Grouping: We choose to group the operations in one SipRound into four separate stages, such that each of the four internal state variables are written to once in each stage and never accessed in the same stage again after being written. We present dependency groupings for a full SipRound in Table 2.3 below.

Stage 1	**Stage 2**	**Stage 3**	**Stage 4**
$a0 = v0 + v1$	$b1 = a1 \oplus a0$	$a2 = b2 + b1$	$v1 = a1 \oplus a2$
$a2 = v2 + v3$	$b3 = a3 \oplus a2$	$a0 = b0 + b3$	$v3 = a3 \oplus a0$
$a1 = v1 << 5$	$b0 = a0 << 16$	$a1 = b1 << 13$	$v2 = a2 << 16$
$a3 = v3 << 8$	$b2 = a2$	$a3 = b3 << 7$	$v0 = a0$

Table 2.3: SipRound Operations Grouped by Dependencies into Four Pipeline Stages

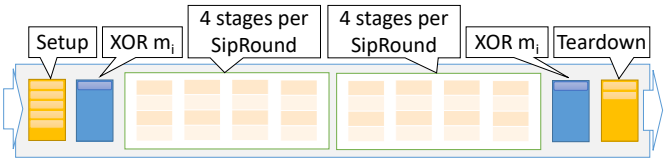


Figure 2.4: We run two SipRounds per switch pipeline pass, with each SipRound costing four stages.

Using these dependency groupings, we can implement k SipRounds in $k \times 4$ stages, with an additional s stages for setup and teardown. If the total pipeline length is S stages, we can perform $k = \lfloor (S - s)/4 \rfloor$ SipRounds per pipeline pass. Then, with a given input string size of M bytes, we require the number of passes given by p:

$$p = \left(\frac{M}{4} \times c + d \right) / k \tag{2.4}$$

Our implementation performs $k = 2$ SipRounds per pipeline pass, as illustrated in Figure 2.4. With $M = 16$ bytes for an input string and the recommended security parameters $c = 2, d = 4$, we need a total of $p = (\frac{16}{4} * 2 + 4)/(2) = 6$ pipeline passes to complete all $(2 + 2 + 2 + 2) + 4 = 12$ compression and finalization SipRounds.

Packet Recirculation: Since we cannot complete all the SipRounds required by HalfSipHash within one pipeline pass (for a 16-byte input, 6 pipeline passes are needed), we utilize the programmable switch's packet *recirculation* feature, which sends the packet to special recirculation ports that simply bounce the packet back to the start of the ingress pipeline. Before a packet is recirculated, we add a special metadata header to the packet to

store internal states and the current round number. We also save the original output port in this metadata header, so the packet can be routed to the output port at the end of the HalfSipHash computation. We use match-action tables with exact match rules to define high-level logic per recirculation pass and perform the correct number of compression and finalization SipRounds.

Ingress+Egress Pipeline: To further optimize our implementation, we design a variation of our implementation that uses both the ingress and the egress processing pipelines, reducing the number of recirculations by a factor of two. More specifically, given $w = M/4$ words of an M-byte input string, HalfSipHash-2-4 needs to run $2w$ compression rounds and 4 finalization rounds. Therefore, for our ingress-only variant, we need to run $2w + 4$ rounds in total and require $r = (2w + 4)/2 - 1 = w + 1$ recirculations. When we run calculations in both the ingress and egress pipelines, we reduce the number of recirculations needed to $r = (2w + 4)/4 - 1 = w/2$. For example, with a 16-byte input processed by our ingress-only variant, $w = 16/4 = 4$ and we require $r = w + 1 = 5$ recirculations. Given the same input string size, by using our ingress+egress variant, we reduce r to be $r = w/2 = 2$ recirculations per HalfSipHash calculation.

Limited Arithmetic Operations. In addition to being constrained to a limited number of pipeline stages, programmable switches are also restricted to specific types of arithmetic operations. In practice, because of these arithmetic limitations, the SipRound

groupings proposed in Table 2.3 cannot be trivially implemented in the switch within strictly four stages.

Specifically, circular bit shifts are the arithmetic operation that consume too many stages. Our programmable switch does not natively support circular shifts using its Arithmetic Logic Units (ALU) and requires separate intermediate operations. A naive implementation of a circular left shift of n bits would need to first calculate the left and right half of the output using two separate shifts (left shift n bits and right shift $32 - n$ bits), and then combine the two intermediate results via bitwise OR:

$$b = (a << n) \;||\; (a >> (32 - n)) \tag{2.5}$$

Because the arithmetic results are not available within the same pipeline stage for immediate use, the bitwise OR would need to happen in the next pipeline stage, thus consuming two stages for a single circular bit shift. Each SipRound requires six circular left shifts, and using this approach, the number of pipeline stages required quickly spikes so that a full SipRound can no longer reasonably fit into a single pipeline pass.

Slicing for Circular Bit Shifts: Instead of using this two-stage approach, we notice that the P4 language supports slicing semantics to extract a subset of bits from a variable. We can thus calculate the same circular left shift by slicing and concatenating bits of the vari-

able as follows, where n is the number of bits to be shifted and ++ is concatenation:

$$b = a[(31 - n) : 0] \text{ ++ } a[31 : (32 - n)] \tag{2.6}$$

With this implementation, a circular shift can be calculated within a single pipeline stage, without the need to use extra intermediate variables. Thus, we can reduce the number of stages required for a circular left shift, and ensure that all operations in a SipRound can complete efficiently within four hardware pipeline stages.

2.5.3 Securely Computing Cookies

SmartCookie uses the HalfSipHash implementation above to compute and verify SYN cookies in the switch agent. Each cookie computation hashes a 16-byte input derived from connection metadata, requiring $w = 4$ input words and thus $2w+4 = 12$ compression and finalization rounds in total. With two SipRounds per pipeline pass, this requires six passes.

In the final switch-agent design, we integrate cookie hashing across both the ingress and egress pipelines, reducing the number of recirculations from 5 to 2 for a 16-byte input. This overhead is acceptable because cookie generation is required only during connection establishment rather than for every packet in an established flow. As we show in §2.8, even with recirculation, our HalfSipHash-based switch agent returns secure cookies 2.6× faster than the next-fastest defense. Additionally, the recirculation limitations of our prototype

result from the lack of cryptographic building blocks in the network hardware, which can be improved in the future.

Key Rotation. To defend against brute-force attacks, HalfSipHash keys are rotated periodically (e.g., every 5-30 seconds). This reflects our underestimation of the time needed to brute-force a key[19]. To ensure handshakes from clients are not accidentally blocked, cookies computed with an old key are still accepted for a short period after a key rotation.

2.6 Split-Proxy Design

With our secure SYN cookie hash in the data plane, we can now safely offload cookie checks to a high-speed switch. However, we must still tackle the challenge of limited switch memory for handling benign flows. SmartCookie accomplishes this with a split-proxy design, where a switch agent and server agent cooperate to performantly stop attack traffic and correctly handle benign traffic. Our design avoids packet buffering during setup and bypasses sequence number translations at the switch agent, allowing it to *approximately* track verified connections and scale beyond available switch memory.

2.6.1 Switch to Server Two-Way Handshake

Figure 2.5 shows SmartCookie's complete setup procedure for verified connections. In `packets 1-3` SmartCookie switch agent verifies clients by performing a secure SYN

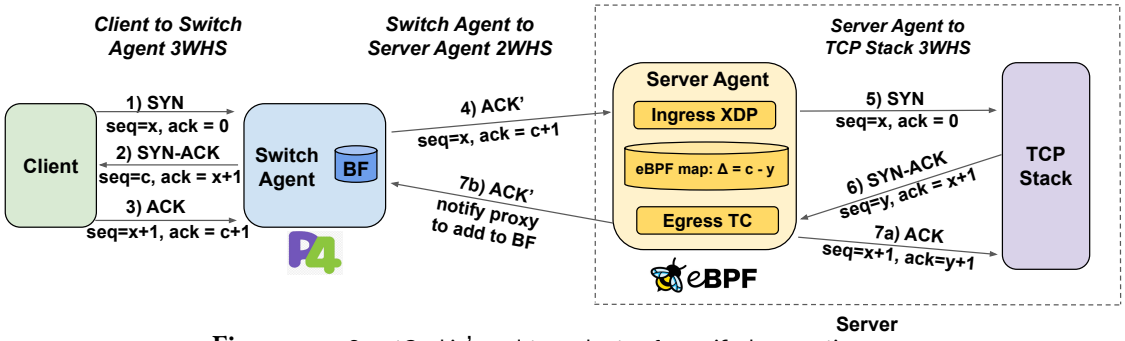


Figure 2.5: SmartCookie's end-to-end setup for verified connections.

cookie check as in the traditional defense. However, under the SmartCookie protocol, the switch agent does not buffer the final ACK of the TCP three-way handshake (3WHS) between the client and switch agent. Instead, the switch agent directly forwards this ACK packet to the SmartCookie server agent with an additional *setup tag*, notifying the server agent to bootstrap the connection setup (`packet 4`). The switch agent also uses this packet to instruct the server agent how to handle the difference in initial sequence numbers (ISNs) that were chosen by the switch agent and the server's network stack.

After receiving this tagged setup packet, the server agent sets up its side of the connection, shown with `packets 5-7a`. The server agent then sends a packet to the switch agent to confirm the connection establishment, completing the custom setup between switch agent and server agent as shown with `packet 7b`. Note that the server agent has been instructed by the switch agent how to handle sequence number deltas, so the client and server see their expected sequence numbers in both directions. This allows the switch agent to step into a passive forwarding role and avoid the expense of sequence number translations

throughout the remainder of the connection.

SmartCookie converts the original 3WHS between switch and server into a custom two-way handshake (2WHS) between switch agent and server agent, notifying the server of the client connection, because at this point the server is still unaware of the client. We note that a 3WHS between switch agent and server agent is undesirable, causing overhead at the switch for buffering packets from the client while the second 3WHS is being conducted. Additionally, even after the end-to-end connection is established, per-packet processing at the switch for sequence number translations is undesirable. Instead, by explicitly informing the server agent of the connection and sending information for sequence number translations, the switch agent can safely forward packets from a verified connection without additional processing.

Handling Setup Latency and Packet Drops. SmartCookie reliably handles more complex scenarios introduced by connection setup latency, reordered packets, or packet drops. Consider the scenario where the 2WHS between the switch agent and server agent is not yet complete, either because it is still in progress or because the setup packet sent to the server agent has been lost. If the client sends additional data packets to the server during this state, the SmartCookie switch agent handles this gracefully by continuing to verify and tag these packets before forwarding them to the server agent (note that the switch agent never buffers client packets). Since the client has yet to receive any packets

from the server, the server-side sequence numbers have not progressed and the client would still pass the cookie check at the switch agent. As long as the client's packets continue to pass the cookie check, the switch agent will tag and forward them to the server agent, and upon receipt of any tagged packet from the switch agent, the server agent will immediately set up the connection and send an explicit confirmation to the switch agent. Upon receiving this confirmation, the switch agent no longer tags any packets from the client and simply forwards packets in both directions.

2.6.2 Redesigned Server Setup

SmartCookie uses *eBPF* to facilitate division of labor between switch and server agents. Our design does not modify the network stack of end hosts, which is particularly beneficial for datacenter deployments where network operators may not control end host TCP behavior (e.g., tenant VMs).

eBPF Primer. eBPF (extended Berkeley Packet Filter) is a powerful and lightweight technology that allows for safe, fast, kernel-level execution of programs directly from user-space, without requiring programmers to rewrite kernel source code[152]. The Linux networking community has utilized eBPF for efficient packet filtering and safe kernel-level execution in many different security applications, including in the DDoS space[120,23,137].

Modified TCP Interface. With eBPF, we redesign the TCP interface between the network and the Linux kernel's network stack. This allows us to offload connection setup and sequence number translations to the server *without* modifying the kernel TCP stack, enabling immediate deployability. Our server agent deploys an eBPF ingress and egress program attached to different kernel hooks (XDP and TC, respectively), performing connection setup with the kernel network stack and sequence number translations on incoming and outgoing packets. The eBPF programs act as a lightweight translation layer that converts custom packets received from the switch agent into proper TCP packets for regular TCP/IP processing by an unmodified kernel.

eBPF Map for Tracking State. eBPF maps can be used to communicate between user space and kernel space, and they are the only way to store and share state between eBPF programs. Maps are implemented as key-value stores, where values are defined by data type and size. The server agent uses an eBPF map to track the connection state of any given flow and to coordinate the behavior of the ingress and egress eBPF programs based on this state. The map key is the 4-tuple connection information (source and destination IP addresses and port numbers). The map value for each key stores the connection state of the connection and the sequence number delta to be applied for that connection. As shown in Figure 2.5, the sequence number delta for a given connection is determined to be $\Delta = c - y$, where c is the switch agent's original ISN (i.e., cookie value) and y is the ISN chosen by the

kernel. At first, the server agent's ingress program extracts c from `packet 4` and stores it in the eBPF map temporarily. Later, the server agent's egress program extracts y from `packet 6` and then calculates and stores the true Δ in the eBPF map.

2.6.3 Compact Data Structure at Switch Agent

To determine which packets require cookie checks, the switch agent (like any standard defense) must maintain a record of verified connections. In prior designs, the proxy keeps a local record of *every* verified connection that has passed the cookie check, using this record to remember the sequence number deltas that must be applied to all packets in the flow. This is an unattractive solution because it requires per-flow state, and memory is a limited commodity in high-speed switches.

Bloom Filters for Approximate State. Since SmartCookie offloads sequence number translations to the server agent, the switch agent can avoid keeping exact state by using a resource-efficient compact data structure called a Bloom filter. The Bloom filter can *approximately* keep track of clients that have passed the cookie check and successfully established a connection with the server. Bloom filters are efficient and powerful approximate data structures, but they come at the expense of a small number of false positives. There are no false negatives in Bloom filters, which is an important feature that ensures benign traffic is never blocked. We note the purpose of the Bloom filter is *not* to stop an adversary's

non-SYN packets from reaching the server, although this does happen as a bonus. Instead, the goal of this design feature is *to not keep per-flow state at the switch agent for verified flows*, reducing strain on the memory-constrained switch.

Implementing Bloom Filters in the Switch Agent. To approximately track the set of benign TCP connections, we use register memory arrays on the Tofino programmable switch to implement a Bloom filter. Since the switch's pipeline only allows accessing one index per array when processing a packet, we implement a Bloom filter variant called the Partitioned Bloom Filter (PBF)[6]. PBF splits its m-bit memory into k separate arrays $M_1[\cdot] \ldots, M_k[\cdot]$, each sized $\frac{m}{k}$, and uses k indexing hash functions $h_1, \ldots, h_k : \mathcal{F} \rightarrow [\frac{m}{k}]$ to map the input key $f \in \mathcal{F}$ (in our case, flow 4-tuple) into locations in the array. All arrays are initialized to zero. To insert a flow f, for each array i, we calculate the index $h_i(f)$ and mark 1 in the corresponding index, i.e., $M_i[h_i(f)] \leftarrow 1$. To query whether flow f has been added, we check the same indices and report positive if they are all 1, i.e., $M_i[h_i(f)] == 1, \forall i \in [k]$.

Automatically Cleaning the Bloom Filters. To prevent over-saturation and maintain the accuracy of the filters as time goes on, we can keep multiple Bloom filters at the switch agent in a rotating fashion, where at any given time window T_W, some filters are being written to, read from, or cleaned (i.e., emptied). A connection is considered active

if any one of the Bloom filters reports a positive. For ongoing connections, we add to the most recent filter when we see server-to-client traffic; note that even for uni-directional upload traffic servers continuously send ACK packets to clients. Thus, when a filter is cleaned, ongoing connections automatically get carried over to the active filter.

We also note that T_W should be chosen such that we maintain a reasonable timeout window that reflects the characteristics of the majority of connections in the network[64]. A T_W value that is too small (e.g., a few seconds) will cause idling connections to be closed too early, while T_W values that are too large will require us to save too many active connections in the Bloom filter, causing increased memory usage and potentially higher false positive rates. Shorter T_W windows (also called "aggressive aging timeouts") are the security best practice for preserving stability and performance while under attack, and can be as short as 10 seconds[48,155].

2.6.4 Handling False Positives at Server Agent

The server agent cooperates with the switch agent to perform a last-line-of-defense cookie check on the small fraction of packets that experience a false positive in the Bloom filter.

Identifying False Positives. The server agent knows to perform this cookie check on any packets that are not already part of an ongoing connection at the server and do not have a setup tag from the switch agent. We note that SYN packets are always stopped and

handled at the switch agent, so the server agent only handles benign traffic and non-SYN packets (either adversarial or benign) that have triggered a false positive in the Bloom filter. If the false positive was the result of an adversary's non-SYN packet (e.g., if the adversary was attempting to perform an ACK flood), it would fail the cookie check at the server agent and get dropped (§2.7.2). However, if the false positive was triggered by a benign client (e.g., with the final ACK packet of the 3WHS between client and switch agent), then the packet would pass the cookie check at the server agent and the server agent would set up the connection with the kernel network stack.

Key Management. In existing SYN cookie defenses, only one party computes and verifies cookies. However, since both the switch and server agents verify cookies, they must share the same SYN cookie computation scheme so that cookies are consistent across the system. In other words, both switch and server agents must share the same hash functions and secret keys used to compute cookies; the network operator acts as the central controller for managing these keys across the system, installing the same initial key and performing periodic key rotation for both the switch and server agents. Furthermore, to ensure consistency when using timestamps to generate and verify cookies, SmartCookie relies on the network operator to synchronize clocks across the switch and server agents. We also note that on the server side, secret keys are only visible to the eBPF switch agent module and are not visible to applications or tenant VMs running on the server.

2.7 Security Analysis

We discuss possible attacks from adaptive adversaries against our system itself (above and beyond our initial SYN-flooding attack vector), and explain how we address them.

2.7.1 Attacks on the Cookie Check

Replay Attacks. When a cookie is first computed at the switch, it is computed with respect to the current epoch of time. When cookies are recomputed and verified, they are only accepted if they are from the current or immediately preceding epoch. While the Linux kernel uses 1 minute epochs [103], we choose 1 second epochs instead, to reflect round-trip times (RTTs) commonly seen on the modern Internet. This ensures that SmartCookie only accepts cookies returned within acceptable RTT delays, while defending against replay attacks with older cookies. For consistency, the cloud provider deploying SmartCookie should ensure that clocks are synced (e.g., NTP) between the switch agent and the server agent.

Cookie Forging and Key Recovery. An adversary can send probe packets to the switch agent and observe the returned cookies in order to crack the hashing mechanism and forge her own cookies; formally, this is performing a Chosen Plaintext Attack (CPA) against our hash function. The security of SmartCookie against cookie forging can be

directly reduced to the security of the underlying HalfSipHash family of hash functions. Assuming the security properties of HalfSipHash, the adversary cannot recover the key without a brute-force search [19]. To make brute-forcing ineffective, SmartCookie rotate the key periodically (e.g., every 5-30 seconds). Thus, with the pseudorandom properties of HalfSipHash, the adversary's best attack strategy is reduced to simply guessing the cookie.

Lucky Cookies, and Saving TCP Options. The Linux kernel's default SYN cookie stores a quantized Maximum Segment Size (MSS) and discards all other TCP options. Our switch agent similarly encodes the MSS in cookies, for the server agent to later decode. This approach leaves 24 bits of entropy in the cookie, meaning that each adversary's attack packet has a negligible probability ($1/2^{24}$) of "luckily" guessing the right cookie by chance. If desired, it is possible to use 3-4 more bits of the cookie to save a few more TCP options, in exchange for slightly reduced entropy.

2.7.2 Attacks on the Bloom Filter

ACK Floods and TTL Expiry Attacks. The Bloom filter used by our switch agent to track verified connections has a small probability of reporting false positives; an adversary can send many ACK packets with randomized source IP addresses and port numbers, and a small fraction of these may trigger a false positive and be forwarded to the server. We refer to this as the ACK-flooding attack, and evaluate the performance of SmartCookie

against such attacks in §2.8. We note that these false positive ACK packets will be silently dropped by the server agent after failing its last-line-of-defense cookie check (§2.6.4). We also filter TTLs of non-SYN packets that arrive at the switch agent, hiding from an adaptive adversary seeking to perform a TTL expiry attack whether her packets were dropped at the switch agent or at the server agent[132]. This prevents the adversary from getting feedback on which attack packets successfully triggered false positives. Thus, she cannot tailor attack packets for the Bloom filter's internal structure to achieve a higher false positive rate.

Opening Many "Legitimate" TCP Connections. The false positive rate of Bloom filters depends on the number of connections inserted. In our threat model (§2.3.1), adversaries have access to a moderate number of compromised devices to launch *asymmetric* SYN-flooding attacks, but not enough to mount successful *symmetric* attacks, such as TCP connection floods. Connection-flooding adversaries would need to establish many "legitimate" TCP connections to raise the Bloom filter's false positive rate before launching a SYN-flooding or ACK-flooding attack, requiring significant resources (e.g., machines, memory, packet processing). Nevertheless, we note our filters are periodically cleaned and designed to achieve a reasonably low false positive rate under realistic traffic conditions (§2.6.3). Even with high rates of Bloom filter pollution, SmartCookie still blocks 100% of SYN floods in the switch and is highly resilient under ACK floods (§2.8.4).

2.8 Evaluation

We evaluate SmartCookie by running several experiments on a hardware testbed. We 1) demonstrate the performance of SmartCookie for securely computing SYN cookies with HalfSipHash in the switch data plane, compared to other approaches, and 2) show the overall performance of SmartCookie in comparison to Jaqen [105], the state-of-the-art switch-based SYN-flooding defense. We note that we owned all testbed infrastructure, and attack traffic was only directed to dedicated testbed servers, raising no ethical issues.

2.8.1 Experiment Setup

Prototype Implementation

SmartCookie's switch agent is implemented in P4 targeted for an Intel Tofino Wedge100-32BF programmable switch, with approximately 1000 lines of code. The server agent is implemented with eBPF. For additional implementation details, we refer interested readers to the published paper's appendix [173]. We have released our prototype source code on GitHub[1]. We also run prototypes of Jaqen SYN Cookie Proxy Mode 1 and Mode 2 [105], based on source code obtained from the authors.

[1] https://github.com/Princeton-Cabernet/p4-projects/tree/master/SmartCookie

Testbed

The testbed consists of four servers and an Intel Tofino Wedge32X-BF programmable switch. Two machines act as adversaries, each with a 20-core Intel Xeon Silver 4114 CPU and a Mellanox ConnectX-5 2x100Gbps NIC, generating attack traffic using DPDK 19.12.0 and pktgen-DPDK. Two other machines act as server and client, each with 8-core Intel Xeon D-1541 CPUs and Intel X552 2x10Gbps NICs. All machines run Ubuntu 21.10 with kernel v5.13.0, and the eBPF programs are built with BCC v0.24.0 and Clang v13.0.0-2. The server machines are all connected to the switch via Direct Attach Copper (DAC) cables, with under 0.1ms ping latency (round-trip time) between any pair of machines.

Traffic Workload

Launching Realistic Client Traffic. To simulate a real-world setting with realistic benign traffic loads, we write customized client and server Go scripts to replay CAIDA anonymized Internet traffic trace 2018 [35]. We did not modify the Linux kernel's default TCP retransmission behavior.

Estimating the False Positive Rate. Our switch agent implementation uses Partitioned Bloom Filters with $k = 3$ arrays of 2^{20} bits each, with total memory size $m = 3 \times 2^{20}$ bits. For our experiments, we choose a connection timeout window T_W of 15 seconds for

Bloom filter cleaning (§2.6.3), which reflects the default keepalive timeout of many applications[64]. We choose a 15-second trace window of benign traffic to match T_W, with $n = 579,600$ flows. Our trace represents heavier flow loads than the average from CAIDA trace statistics (386,000 flows over the same time window)[36]. Thus, we generously estimate our false positive rate as:

$$F_p(n, m, k) = \left(1 - \left(1 - \frac{k}{m}\right)^n\right)^k \approx 7.66\%. \tag{2.7}$$

We replay each flow as an HTTP request in real time (approximately 38,500 requests per second), where the request starting time corresponds to the timestamp offset of the flow's first packet. We also measure the number of attack packets received at the server with this setup and verify that the measured false positive rate matches our expectations. Meanwhile, since Jaqen's prototype uses a smaller Bloom filter ($2^{16} \times 4$), we replay fewer flows (48,800) in Jaqen's experiments, such that the defense exhibits the same false positive rate of 7.66%. Since the number of connections in both experiments exceed the number of available ports, we add 65,536 IP addresses to both client and server; we use the connection 4-tuple as keys for Bloom filter lookups for both SmartCookie and Jaqen.

More generally, under different traffic conditions, time windows, and number of connections, we can use Equation §2.7 to estimate corresponding false positive rates. We also note that our current implementation did not exhaust the memory available on the Tofino

1 switch, and we can build larger Bloom filters to support more connections, trading off more memory for a reduction in the false positive rate. Other switch models (e.g., Tofino 2) also provide more onboard memory to further increase Bloom filter size and lower false positive rates.

Generating Attack Traffic. We run pktgen-DPDK on the adversary machines to generate SYN floods with randomized source IPs and ports. Due to false positives, Jaqen will allow 7.66% of SYN-flooding traffic to reach the server and trigger a *connection setup and kernel SYN cookie computation*. Meanwhile, since our switch agent handles 100% of SYN packets without passing any to the server, we run a separate experiment where we subject SmartCookie to ACK flood traffic (§2.7.2), 7.66% of which will reach the server and trigger *only eBPF SYN cookie verification* at the server agent.

2.8.2 Hashing Throughput

Under the experiment setup described above, we compare SmartCookie's performance against kernel-based SYN cookies and XDP-based cookies. Since prior work proposed running AES, a secure encryption algorithm, on Tofino switches[40], we also implement and benchmark a variant of SmartCookie switch agent using AES to compute cookies.

Measuring Throughput. We measure the maximum attack rate in Mpps each defense can handle *before any packet loss*. Since our benchmark performs one hash calculation

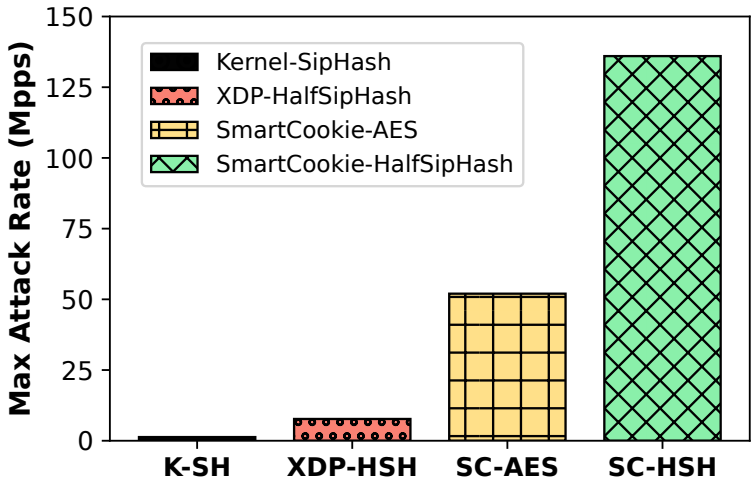


Figure 2.6: Throughput. SmartCookie-HalfSipHash defends against attacks *without packet loss* until a rate of 136.9Mpps, outperforming the next fastest defense by 2.6x.

per SYN packet, we effectively measure maximum *hashing* throughput.

Results. As shown in Figure 2.6, SmartCookie-HalfSipHash significantly outperforms all other defenses, achieving a throughput of 136.9Mpps on high-speed switch hardware. This is *two orders of magnitude greater* than the throughput achieved by the software-based kernel defense, which uses SipHash and can only serve 1.3Mpps before the server's CPUs are exhausted. Using XDP to compute cookies can bypass the overhead of the kernel network stack and achieve 6x speedup (7.3Mpps), but this is still much slower than SmartCookie-HalfSipHash. Meanwhile, although SmartCookie-AES is faster than XDP, it requires more recirculations and only achieves 52 Mpps; SmartCookie-HalfSipHash outperforms it by 2.6x.

We note that Tofino switches have dedicated recirculation ports, *which operate with-*

out affecting capacity on other ports. Our prototype switch agent achieves 136.9Mpps using a single Tofino 1 switch under its vanilla setup (pre-configured ports with 200Gbps recirculation throughput). To achieve even higher throughput, we can simply load balance between multiple switches to multiplex their throughput, or configure the switch to convert unused physical ports into extra recirculation bandwidth. For example, as a back-of-the-envelope calculation, using 20 recirculation ports, we could achieve 2Tbps recirculation throughput and serve roughly 1.4 billion requests per second. We also note that while repurposing additional ports for recirculation would reduce overall switch capacity, it will not affect latency. Finally, other switches exist (e.g., Tofino 2) with higher per-port throughput and more pipeline stages. Such switches, along with future hardware with native cryptographic support, can further reduce recirculation needs, additionally boosting throughput.

2.8.3 Latency

In this experiment, we launch both benign and attack traffic against a server as described in §2.8.1, and measure the end-to-end latency of benign application traffic while the server is under attacks of different magnitudes. We compare SmartCookie against the vanilla kernel-based SYN cookie defense and Jaqen's two SYN proxy modes. SmartCookie's design is transparent to clients and does not cause connection reset, leading to 48%-84% lower latency than Jaqen.

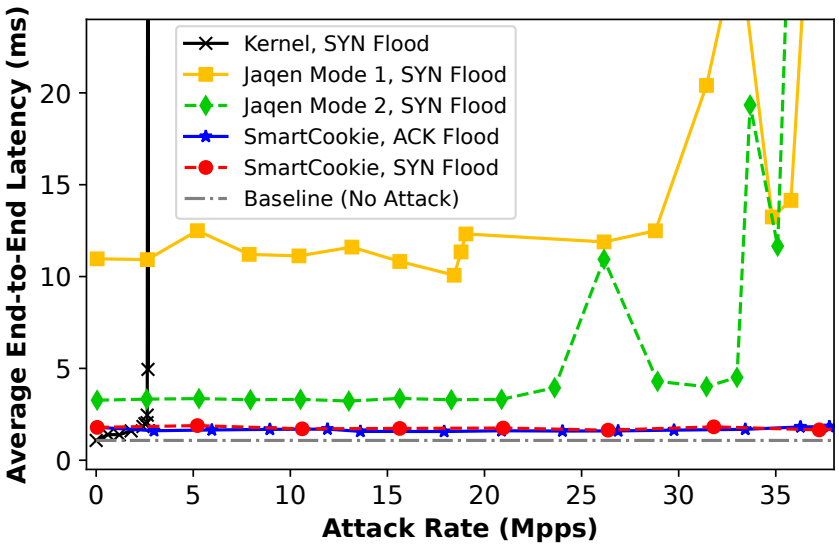


Figure 2.7: Latency. SmartCookie reduces latency by 48-84% compared to Jaqen, protecting client performance.

Measuring Latency. We measure the end-to-end application latency by initiating HTTP requests from the client to the server, with response size of 14.5KB (corresponding to the average flow size in the CAIDA 2018 trace). For each attack rate, we send 30 requests and calculate the average latency. Since Jaqen's Proxy Mode 2 triggers an application-layer reset, most applications (`curl`, `wget`, etc.) will wait for at least one second before retrying the connection. This timeout is an unreasonably large penalty for benign clients, and so we write a customized Go client script that immediately retries after a reset, waiting for only 1ms between connection attempts. This showcases the best possible performance of Jaqen, as it avoids the long default reset timeout, but we note that reconfiguring client applications is not always possible.

Results. As shown in Figure 2.7, SmartCookie has consistently low end-to-end latency (1.71ms), a 48%-84% reduction compared to Jaqen, even with the reconfigured fast retry client that bypasses Jaqen's default 1-second timeout. Jaqen Mode 1 incurs a reset and one additional round trip, with a minimum latency of 11.12ms. Jaqen Mode 2 incurs two additional round trips and an application-layer retry, requiring at least 3.31ms for end-to-end setup with our fast 1ms retry client and *over one second* for default TCP applications that have not been reconfigured. Meanwhile, SmartCookie's latency, which is only 1.71ms, is close to the baseline latency of the vanilla kernel without any attack (1.08ms).

2.8.4 Server CPU Usage

Using the same setup from §2.8.1, we replay trace-based benign traffic while launching attack traffic at increasing rates. We measure the server's CPU overhead across various defenses. SmartCookie has *no overhead* for SYN floods and reduces overhead for ACK floods by 33-36% compared to Jaqen.

Measuring CPU. To measure CPU overhead, we use the `perf stat` command to read CPU performance counters and collect the number of instructions executed per second for each CPU core, taking the average across cores. We also repeat each measurement ten times and take this average. For more stable results, we turned off frequency scaling (Turbo Boost) and fixed all cores' frequency at 2.1GHz.

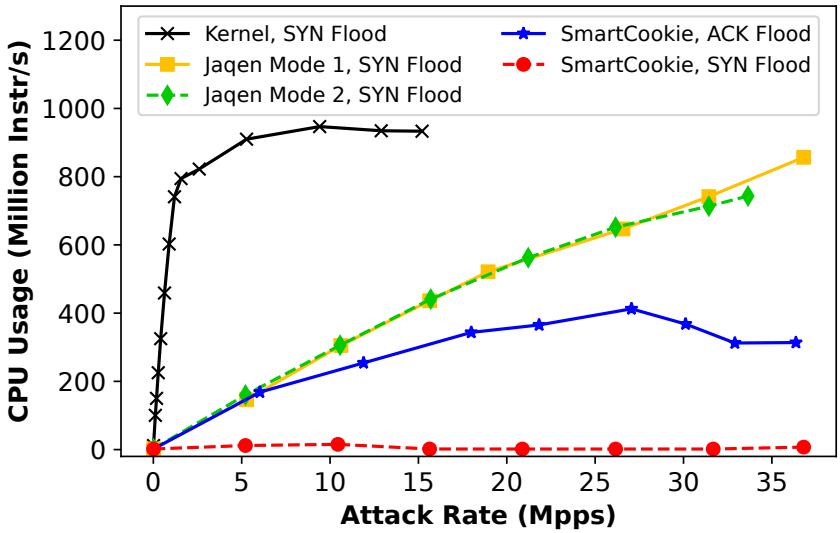


Figure 2.8: CPU Usage. SmartCookie has *no* server CPU overhead against SYN floods and reduces overhead against ACK floods by 33%-36% compared to Jaqen.

Results. Both modes of Jaqen exhibit similar server CPU overhead, as they both allow the same rate of false positive SYN packets to reach the server, where the defense degrades to SYN cookie computation on server software (Figure 2.8). We note that SmartCookie switch agent processes 100% of SYN packets directly on network hardware, completely protecting server CPUs from resource exhaustion for SYN cookie computation. With SmartCookie, servers experience *no CPU overhead during SYN floods*. Thus, in order to see the effect of false positives on CPU usage with SmartCookie, we also measure the overhead of the server agent handling map lookup and cookie verification for false positive ACK packets (§2.6.4): SmartCookie under an ACK flood exhibits approximately 33%-36% lower CPU overhead for the same attack rate and same false positive rate as Jaqen under a SYN flood. This is because Jaqen is a purely switch-based defense that degrades to

	SRAM	TCAM	HashUnit	Instr.
Jaqen Mode 1 (CRC)	4.6%	0.0%	19.4%	4.9%
Jaqen Mode 2 (CRC)	4.6%	0.0%	19.4%	6.3%
SmartCookie-AES	13.8%	0.1%	30.6%	9.4%
SmartCookie-HSH	6.8%	1.4%	56.9%	9.9%

Table 2.4: Resource usage on programmable switch.

	SRAM	TCAM	HashUnit	Instr.
tna_simple_switch.p4	6.3%	9.0%	0.0%	9.6%
+SmartCookie	12.6%	11.5%	54.2%	18.0%
switch.p4	33.6%	31.6%	19.4%	13.5%
+SmartCookie	40.1%	37.5%	73.6%	20.3%

Table 2.5: Adding SmartCookie to complex P4 programs.

the vanilla kernel's defense on false positives, which begins to suffer from attack rates as low as 1.3 Mpps. Meanwhile, SmartCookie is designed as a split-proxy system that cooperates with an XDP-based server agent for false positives. XDP operates on raw packets early in the kernel stack before any socket buffer is allocated, lowering the overhead significantly compared to the kernel defense.

2.8.5 Switch Resource Usage & Compatibility

In Table 2.4 we report SmartCookie's utilization of switch hardware resources with two variants of hashes (HalfSipHash and AES), as reported by Intel's P4i tool. SmartCookie has a trivial footprint for important shared resources (TCAM:1.4%, SRAM:6.8%). Notably, the HashUnit usage is highest (57%), but HashUnits are used less by other network functions. We also report resource utilization of Jaqen SYN Proxy (Modes 1 and 2), which

use CRC as its cookie hash.

Furthermore, we successfully integrated SmartCookie-HalfSipHash into two feature-rich, complex P4 programs: `tna_simple_switch.p4` and `switch.p4`, demonstrating SmartCookie can co-exist with other sophisticated network functions. Table 2.5 presents resource utilization metrics for each base program and the variant with SmartCookie added, highlighting SmartCookie's efficient footprint and compatibility with other switch functions.

2.9 Discussion

Routing Considerations

All traffic between clients and servers traverses a network switch (i.e., the edge switches are a 'cut' of the network between clients and servers). We envision our defense deployed in a provider network with multiple edge switches. In a distributed defense with an upstream switch it is not guaranteed that all traffic will pass through the *same* switch in both directions (asymmetric routing) and traffic flowing in a single direction may also traverse different switches (routing changes). Asymmetric routing is very common due to hot-potato routing, a phenomenon where networks seek to pass traffic as quickly as possible out of their borders, resulting in traffic for one connection not taking the same path in both directions[119]. Routing changes, while less common at the connection level, can still occur and in

particular affect longer-lived connections [119].

With an upstream defense, traffic from an ongoing flow could traverse a switch that did not perform the initial cookie check, motivating the need to be robust to both asymmetric routing and potential routing changes. This must be handled gracefully so the client can avoid additional verification or connection disruption. SmartCookie handles asymmetric routing by default with several clever design decisions, and it can handle routing changes using two high-level approaches, each with their own tradeoffs. Cooperating switch agents can maintain synchronized Bloom filters with state for all verified connections. Alternatively, Bloom filters across individual switch agents can dynamically adapt to routing changes using a packet sampling approach.

Handling Asymmetric Routing. The SmartCookie protocol has removed the need for the switch agent to remain actively involved (*i.e.,* with sequence number translations) in a connection once it has been established. This ensures the switch agent need only *forward* return packets, especially since this traffic is coming from servers and can be trusted without undergoing verification. Thus, once the setup phase is complete, *any* switch in the provider backbone can safely forward the rest of the return traffic, supporting asymmetric routing and direct server return (DSR), regardless of the extent of asymmetry.

During connection setup, however, the switch agent must see the return traffic from the server to properly add the connection to its local record of verified connections. To

accomplish this, the SmartCookie server agent ensures response traffic is routed through a single switch agent during setup by using loose source routing (LSR) to explicitly route response traffic from the server to the client through the *same* switch agent, until the switch agent has created a local record. In practice, when the switch agent sends a tagged packet to the server agent, it includes a switch agent identification (i.e., the IP address of the switch) that the server agent can use for routing relevant return packets. The server agent knows when the switch agent has updated its local record since the switch agent will stop tagging packets once the update is complete.

Handling Routing Changes. Routing changes are less common at the TCP connection level, especially when flow-aware load balancing (e.g., ECMP) is used instead of packet spraying[82]. Still, our switch agent must gracefully handle routing changes caused by configuration updates, equipment failures, or other unexpected reasons. We consider a scenario where a client has passed the cookie check at a given switch agent, `SwitchAgent_1`, and has been added as a verified connection to its local Bloom filter (BF), but has some packets routed to the server through a different switch agent, `SwitchAgent_2`, which does not have records of the connection. In this case, SmartCookie should ensure that a verified client would not get penalized by experiencing connection disruption (e.g., packet drops or connection reset) at the new switch agent.

BF Synchronization: For provider backbones that expect routing changes more regularly

(e.g., with packet spraying), one solution could be to replicate BF state across the different switch agents, maintaining a copy of the set of all verified connections at each switch agent in a fault-tolerant manner[93]. However, if routing changes are expected infrequently (e.g., all connections are short-lived), synchronizing BF state could be avoided, as this would fill up the BFs unnecessarily and degrade the membership-reporting accuracy of the filters.

Packet Sampling: To avoid synchronizing state across BFs, the SmartCookie protocol could allow individual BFs to dynamically adapt to routing changes. To accomplish this, the switch agent could probabilistically allow a small sampling of non-SYN packets through to the server agent even if the packets are not in the BF and have not passed the SYN cookie check. By performing stateless rate limiting with probabilistic sampling, SmartCookie would trade off the risk of a routing change causing benign traffic to be dropped and the risk of allowing attack traffic through to the server agent.

More concretely, we consider a case where an adversary sends attack traffic with the same connection 4-tuple (spoofed from multiple vantage points) in order to cause traffic to enter the provider backbone through more than one switch agent. If the attack traffic consisted of SYN packets, each of the switch agents would initiate the SYN cookie check and the attack would be stopped at the network edge. For non-SYN packets, the switch agent would probabilistically let through a small number of unverified packets, placing a special tag on these packets before forwarding them to the server agent.

Upon receiving a packet with this special tag, the server agent would check to see if the packet was part of an active connection. If it was, the server agent would use LSR to respond to the specific proxy that sent the packet, signifying all is well and the switch agent can add the connection to its BF. However, if the packet was not part of an active connection at the server, the server agent would simply drop the packet and no further action would be taken. We note that this design requires minimal effort from the server agent, as it would simply perform connection lookups and would *not* perform any cookie checks for these special packets.

Serving Tenants Transparently

SmartCookie does not modify the server's network stack, and can serve unmodified tenant VMs running on servers. From the tenant's point of view, the TCP protocol is unchanged. Today's high-performance VM hypervisors and container hosts often run specialized software switches (e.g., eBPF-based Cilium [46]) to handle tenant traffic, and SmartCookie's server agent can be integrated into these software switches.

Handling Other DDoS Attacks

Although SmartCookie was explicitly designed for large-scale SYN-flooding attacks, by default it also handles other TCP-based volumetric attacks (e.g., ACK floods, SYN-ACK floods, RST/FIN floods), quickly dropping attack traffic on behalf of the server. We believe

our design can also be generalized to UDP-based volumetric attacks, with mechanisms like those proposed in [163].

Transport Protocols

We note that SmartCookie makes assumptions about TCP and must be upgraded when clients use new features, like MPTCP. To avoid protocol ossification, SmartCookie should not be applied by default on all traffic. Instead, it should be an opt-in feature for tenants enabled only during attacks, like the Linux kernel's SYN cookie defense. Future servers and tenants using newer transport protocols would require updated designs; split-design defenses supporting newer protocols (e.g., MPTCP) and connection-oriented UDP traffic (e.g., QUIC) are interesting future works.

Header Field Compatibility and MTU

Setup tags added to packet headers are only visible within the cloud provider's internal network, between switch agent and server agent. Any such modifications are removed before unmodified packets are passed to the server's unmodified TCP stack. SmartCookie mostly tags handshake packets during connection setup, when packets are only 40 - 60 bytes. However, in special cases (setup packets dropped or routing changes) SmartCookie might tag a client's data packets, so the internal network's MTU (Maximum Transmission Unit) should be roughly 10-20 bytes larger.

Further Improvements

Our eBPF server agent is independent of the kernel network stack, enabling direct deployability without kernel updates. However, the server agent needs to pay a small overhead for translating sequence numbers on every TCP packet. Without changing the trust model or requiring tenant VMs to have access to secret keys, a tighter integration with the Linux kernel of trusted physical servers can eliminate this performance penalty. Specifically, the server agent can explicitly instruct the kernel to choose the desired initial sequence number (ISN) and synchronize numbers between the switch and server.

Synchronizing the Initial Sequence Number. An unmodified Linux kernel chooses its ISN at random. By integrating the server agent 2WHS with the kernel's TCP connection setup, we could explicitly instruct the kernel to adopt the ISN chosen by the switch agent. This minor change in the kernel network stack would remove the need to perform sequence number translations in the server agent altogether, further improving its performance and automatically allowing the switch agent to avoid keeping per-flow state.

Avoiding Duplicate Per-Flow State at Server Agent. We also note that currently the eBPF-based server agent maintains per-flow state for all TCP connections in an eBPF map, duplicating the kernel-maintained list of all TCP sockets. This leads to a small but non-negligible memory overhead. Future implementations of the server agent could

query the connection state from the kernel's list of active TCP connections directly. This would allow the server agent to become stateless after connection setup, acting as an extremely lightweight module that only handles the 3WHS with the TCP stack and does not keep per-flow state.

Offloading the Server Agent. We can push the server agent to run directly on a NIC that supports eBPF hardware offloading, removing the overhead of running on the server's CPU. We can also run the server agent on a SmartNIC (e.g. NetFPGA), or even on a top-of-rack (ToR) switch in front of the server. These techniques would allow the server's CPU to be dedicated to only running application logic.

2.10 Related Work

SYN Cookie Defenses. State-of-the-art SYN cookie defenses, including standard practices such as DDoS scrubbing centers, struggle to capitalize on the unique capabilities of programmable switches. They either have the data plane do *too much* (e.g., performing the complete TCP handshake or keeping per-flow state) [105,177] or *too little* (e.g., simply forwarding attack traffic to a server for software-based packet-processing) [58,57]. These works miss opportunities to refactor the server, under-optimize switch resource usage, incur performance penalties on benign clients, and most importantly create vulnerabilities in the defense with insecure hashes.

The most closely related such work is Jaqen [105]. Unlike Jaqen, SmartCookie uses a cryptographically robust hash that provides strong security guarantees and does not sacrifice performance for scalability. [130] also proposed a SYN cookie proxy, but their proxy must similarly keep per-flow state and is implemented in software [30]. SmartCookie overcomes strict resource constraints to efficiently run SYN cookies directly on switch hardware.

Split Functionality. Poseidon [177] presents a two-part DDoS defense: a switch component on hardware and a server component running in software. However, Poseidon requires the majority of the defense to reside in software, which is orders of magnitude slower at packet-processing than switch hardware. More importantly, Poseidon's SYN cookie proxy uses an insecure hash, and it must also keep per-flow state. SmartCookie overcomes these limitations and presents a novel split proxy that preserves switch resources, optimizes performance, and computes cookies with a *secure* hash.

XDP. GateBot [23] is a DDoS defense system developed by Cloudflare that drops traffic based on iptable rules implemented with XDP. Recently, Google also used XDP to implement SYN cookies, taking advantage of XDP's early execution hook to bypass the kernel TCP stack [120]. SmartCookie provides an order of magnitude higher performance than XDP-based SYN cookies, stopping all SYN flood traffic in switches at the network edge and only relying on XDP-based cookies to handle a small number of false positives.

Cryptography in the Data Plane. Prior works have explored cryptographic functionality in the data plane, although some targeted software models instead of hardware environments. Scholz et al.[131] implemented prototypes of cryptographic hash functions, including SipHash, targeted for CPU backends, NICs, and FPGAs.[40] and[172] first implemented the AES cipher and HalfSipHash for Tofino switches. Our work leverages such cryptographic building blocks to design and implement a complete split defense system with end-to-end evaluation results. NeoBFT[148] further optimized HalfSipHash on switches by computing multiple message signatures in one batch. PINOT[163] implemented a lightweight 2-round Even-Mansour (2EM) cipher for switches for privacy-protecting IP address encryption; however, their implementation uses fixed permutations that cannot be quickly rotated, which introduces a potential risk for brute-force attacks.

General In-Network Defenses. Previous works used software-defined networking (SDN) to implement SYN-flooding defenses, outside of SYN cookies[58,110,136,108,125,60]. Approaches include rate-limiting based on an upper-bound threshold of SYN packets[108], enforcing whitelists or blacklists based on completion of the TCP handshake[110,60], and using switches to identify and steer suspicious traffic to software platforms for further handling[58]. More generally, other recent works proposed switch-based defenses against link-flooding attacks[181] and pulse-wave DDoS attacks[72]. Surveys of switch-based DDoS mitigations and other switch-based network security applications are presented in[43,44].

2.11 Conclusion

SmartCookie is the first SYN-flooding defense to provide cryptographically secure SYN cookies on high-speed switches. With its novel split-proxy architecture and layered hardware-software co-design, SmartCookie remains robust against adaptive adversaries while scaling to large attack volumes, without disrupting benign traffic or degrading performance. More broadly, this chapter shows that carefully partitioning functionality across complementary hardware and software layers can outperform purely hardware designs, and thus yields a general design principle for building scalable edge defenses on modern programmable networks under realistic resource constraints.

Part II

Route Control via

Inter-Edge Cooperation

3

Exposing and Securely Measuring Alternative Interdomain Paths

In this chapter, we present TANGO, which takes a first step toward enabling *route control* through selective cooperation among edge networks. TANGO allows cooperating edges to expose additional BGP-compliant interdomain paths beyond the default route, without

relying on third parties or changes from the Internet core. We show that Tango edges can jointly (*i*) expose more BGP-compliant wide-area paths via coordinated BGP advertisements; (*ii*) collect fine-grained, trustworthy telemetry using cryptographically-protected custom headers; and (*iii*) dynamically reroute traffic in the data plane. Tango innovates in both the control and data planes, and runs on a programmable switch or in eBPF. In this way, Tango lays the foundation for route control *between* multiple edges, extending the dissertation's edge-centered framework beyond ingress control *within* a single edge. Our Internet-scale experiments uncover rich path diversity, exposing paths that outperform the default BGP path 75-100% of the time for 20 edge pairs across multiple continents, while reducing latency by up to 39% compared to the default.

3.1 Introduction

Modern networked applications, from self-driving cars to online gaming and video conferencing, have strict requirements of high reliability and low latency[75,5,91,115]. To satisfy these needs, hypergiants continually expand their private network infrastructure closer to the edge, effectively optimizing client experience. For instance, Google not only operates multiple Points of Presence (PoPs) globally, connecting data centers to the rest of the Internet via peering, but also partners with ISPs to deploy Google-supplied servers inside the ISP networks[83]. Similarly, Azure proposed deploying physical infrastructure inside enter-

prise premises to optimize ingress traffic[95]. Edge networks such as enterprises and small clouds, however, are unable to continuously expand their infrastructure and are forced to resort to alternatives such as Network-as-a-Service (NaaS)[15] from global cloud providers, outsourcing their traffic management while inheriting third-party practices and security/privacy policies. This private-WAN trend leads to increased industry consolidation, benefiting larger companies and well-served regions while leaving smaller edge networks with limited negotiation power, reduced growth opportunities, and increased vulnerability to outages[170,89]. The natural question arises: *Is moving to private infrastructure the only way to meet growing application requirements, or can the public Internet rise to the challenge?*

To answer this, we must first understand the obstacles preventing an edge network from extracting more performance and route control in today's public Internet. Edge networks have very limited available path diversity. BGP (*i.e.,* the default Internet routing protocol) selects a single path per destination prefix based on crude (often performance-unaware) criteria[180]. While multi-homed ASes can optimize the first hop of their path[68,4,3], they are unable to tap into the Internet's full path diversity without collaboration from the Internet core. SD-WAN solutions[15,16,159] that combine multi-homing, overlay routing, and multicast techniques are still limited to BGP-default paths. Even assuming an edge network could forward traffic via adequately distinct paths, identifying performance opportunities requires accurate and trustworthy monitoring, which is impossible in practice for a single

edge. Indeed, measurements at the hosts [116,113] or even at the border of a stub network [79,12] are affected by the performance of both the forward and reverse path and are inflated by the load on the receiver's access network, their hardware, and even the application itself, hiding performance differences between paths. Active probing might avoid some of this noise, but probes can be preferentially treated, hence unreliable [20]. Worse yet, some ASes might attempt to fool any measurement infrastructure to attract more traffic or hide their outages.

To overcome these obstacles, we present Tango, an edge-to-edge route-control scheme that relies on *cooperation* between pairs of edge networks (*e.g.,* enterprises and data centers). We observe that collaboration between edge networks occurs naturally in today's Internet, either among sites of the same organization or between pairs of enterprises, creating many real-world opportunities for Tango. Tango exploits this collaboration to expose multiple routes per destination that are already installed in core routing tables (but not used by BGP) by advertising multiple prefixes for the same destination. While Tango edges cannot explicitly change how on-path routers forward traffic, they can remotely guide the propagation of BGP advertisements for each prefix via surgical use of BGP communities and path poisoning, in an *automated manner and with no prior topology knowledge*.

Tango also performs accurate and trustworthy wide-area monitoring between two edges, another building block towards reliable and real-time route control. Edges can oper-

ate Tango nodes at their border gateways to piggyback telemetry information (metadata) on every packet at the sending edge[92]. This metadata is then stripped away at the receiving edge, before the packet is forwarded to its destination. In this manner, Tango edges obtain accurate *wide-area*, *one-way* measurements unpolluted by reverse path metrics, noisy access networks, or application glitches. Further, relying on metadata makes Tango *protocol-agnostic* (does not rely on TCP semantics) and scalable (does not keep per-flow or packet state[101,176]). Most importantly, since Tango operates over the untrusted public Internet, it provides *trustworthy* telemetry using shared book ciphers and secure OTP-protected route updates directly in the data plane.

Tango has potential for many modern use cases. For example, a cloud provider without its own private WAN can run Tango across the Internet between its data centers for dynamic route control. Emerging online gaming services can establish agreements with remote edge networks to optimize latency and jitter for their customers over the Internet, without investing in on-premise infrastructure or pairing with a cloud provider[178]. Edge networks leveraging federated learning to train models without sharing data with each other or with a cloud provider[21] can use Tango to optimize communication between each participating network and the parameter server, which is often a bottleneck[100] or even security hazard[135]. Finally, datacenters running miners, validators, or decentralized exchanges can use Tango to improve their pair latency and path diversity (hence their performance

and security [128,13,54,153,11,10]), without sacrificing their decentralized nature by moving to a single cloud.

As an end-to-end system, Tango allows edges to optimize interdomain traffic according to their desired objectives, providing them with the means to (*i*) forward their traffic through paths they did not know existed; (*ii*) accurately measure relative loss and delay even in the presence of adversaries; and (*iii*) securely reroute traffic in real-time. This work does not seek to innovate on the *path-selection* algorithm, nor does it make claims on its stability and optimality. In fact, we find that Tango can yield non-trivial benefits for Tango edges in the wild even with a simple control loop, *e.g.,* selecting the path that maintains significantly lower latency for over 100ms. While, in theory, greedily optimizing routes based on local preferences might impact path conditions, we believe that in practice, independent edge pairs are less likely to affect other traffic by congesting links, especially compared to alternative large cloud systems with heavier traffic loads [129,95,171].

To reap all these benefits, two cooperative edges only need to (*i*) deploy lightweight Tango logic at their border gateway, which controls routing between the cooperative edges and (*ii*) have access to a BGP speaker which can advertise a set of prefixes. While Tango is highly deployable, since it can run on either a programmable switch or with eBPF, its modular design further eases adoption. In fact, each or a subset of the components can be independently deployed and directly benefit adopters. For example, Tango's

trustworthy telemetry scheme can be independently used to reliably measure loss and delay, verify service level agreements (SLAs), or detect violations of network neutrality. Similarly, clouds and distributed enterprises can use our Tango's path-finding algorithm alone to expose paths they did not know existed.

In our ethically-conducted Internet-scale experiment between 23 pairs of Tango nodes in globally-distributed Vultr data centers[162], Tango's automated path discovery tool exposed 3-12 distinct paths beyond the BGP default. Interestingly, for 20 pairs, one or more Tango-uncovered paths outperformed the default for 75-100% of the time, with some improving one-way latency by up to 39% (§3.6.1). Meanwhile for 6 pairs, an alternative path improved latency by at least 20% or more for over 10 hours on average. We also estimated with large-scale simulations across 999,000 randomly chosen pairs in the Internet topology that Tango can expose at least two new paths for 98.6% of tested pairs, without collaboration from the Internet core (§3.6.2). Finally, we ran Tango end-to-end between a switch and eBPF deployment on two continents, showing the practicality and performance of our real-time routing control in the wild (§3.7).

3.2 Tango Problem Setting

In this section, we explore challenges of optimizing routing in today's Internet and highlight key requirements for a secure and practical interdomain route-control system.

3.2.1 Challenges of Today's Internet

Lack of Route Control. Despite the rich path diversity of the Internet (§3.6.2), the default interdomain routing choices of an edge network are limited to its direct neighbors. With standard BGP, each AS only exposes one path to each neighbor independently of performance, and so a single-homed network has no choice beyond the *single* BGP route its direct provider offers for each destination IP prefix. Meanwhile, a multi-homed network might only select among very few providers, which can have common bottlenecks, making such solutions limited [12,4]. Any source routing protocol or multipath extension to BGP requires the participation of multiple ASes, making deployment difficult. Similarly, MPTCP only operates once paths are exposed, and it is protocol-specific.

Inaccurate Measurements. Collecting accurate performance measurements that are suitable for comparing wide-area paths is challenging. First, end-to-end measurements are often dominated by issues in the edge network (*e.g.,* wireless interference or local congestion) or on the end hosts themselves (*e.g.,* an overloaded machine) which, in essence, add noise to the wide-area path performance. Second, bidirectional metrics such as round-trip time (RTT)—whether collected by end-hosts or by network devices—are hard to decompose into separate metrics for the two one-way paths. Instead, separate measurements are required for path selection which is naturally one-way. Finally, measurement strategies of-

ten rely on protocol semantics (*e.g.,* TCP sequence and acknowledgment numbers), which do not generalize to all traffic, *e.g.,* QUIC[98], thus reducing the chances of reliable passive measurements or even ignoring the performance of certain protocols.

Untrusted Network. Route control over the public Internet (*i.e.,* an untrusted network) requires consideration of on-path adversaries. On-path adversaries may try to fool the monitoring infrastructure (or any data-driven system[107]) for monetary gains (*e.g.,* to attract more traffic to their path to generate higher revenue, perform traffic analysis, or hide poor performance to avoid SLA violations)[20]. Secure monitoring using cryptography at scale is very challenging in today's networking hardware.

Poor Deployability. The many proposals for optimizing Internet routing are notoriously hard to deploy in practice, creating a pressing need for low-cost and readily deployable solutions. Existing approaches often require core networks to run a new variant of BGP[38], deploy additional overlays[2], or run an entirely new routing protocol[121]. For instance, public overlay networks (*e.g.,* RON[8]) and future Internet architectures (*e.g.,* SCION[121]) require worldwide deployment, extra infrastructure (with associated costs), end-to-end coordination, and overheads for software processing on end-hosts.

3.2.2 Tango Design Requirements

To overcome these challenges in the wide-area setting, we propose Tango, a platform allowing edge networks (*e.g.,* small data centers and enterprises) to optimize interdomain routes. The following constraints drive Tango's design.

Incentive Compatibility (§3.4, §3.5.1). Tango should only expect cooperation from edge networks that actively benefit from routing optimizations (*i.e.,* source and destination networks of exchanged traffic). Thus, it should be transparent and should not rely on support from ISPs or intermediate ASes in the Internet core. In addition, the entry-level investment for individual networks to use Tango should be minimal.

Plug-and-Play Control (§3.4). Tango should enable edge networks to leverage Internet path diversity without requiring them to have multiple providers/peers, knowledge of the wide-area topology or expertise in advanced routing techniques. Observe that private-WAN approaches typically rely on highly connected PoPs and knowledge of the intermediate topology to control routing over available paths[95]. This is impractical for smaller edge networks connected over the Internet, where only incomplete topology approximations are available[37,86]. Moreover, Tango must *automatically* discover and expose paths *without* requiring manual input from the operator.

Accuracy & Timeliness (§3.5.1, §3.5.3). Tango should allow participating networks to accurately measure paths and react in a timely manner to changes, dynamically choosing different routes based on collected performance measurements.

Trustworthiness (§3.5.2). Tango should be robust against adversaries attempting to influence routing decisions by making a path appear more performant than it is. Concretely, we assume adversaries can intercept (and thus modify) packets on at most $n-1$ of n paths, and they can observe (eavesdrop) on any path. Under this threat model, we describe in more detail a few concrete attacks that can be launched.

1. *Manipulation Attacks*: An adversary on path P wants to make the one-way-delay of path P look smaller. The adversary launches the attack by modifying a timestamp t to timestamp $t+d$ which will make its one-way-delay appear to be reduced by d. Similarly, the adversary could modify the sequence numbers to hide loss and cover-up that some packets were dropped in her network.

2. *Replay Attacks*: An adversary is on paths P and Q. The latency from the sender to the adversary along P is shorter than the latency from the sender to the adversary along path Q. Latency from the adversary to the receiver is the same on path P and Q (P and Q may even share all hops after the adversary). At an instant t in time, the adversary sees packets with timestamp $t-p$ on path P and $t-q$ on path q. Since path

P is faster, $t - p > t - q$. The adversary wants to improve the one-way-delay on path q and rewrites the timestamps for path Q to be $t - p$, reducing the delay on Q to the delay of P. Alternatively, the adversary could have a passive tap on path P.

3.3 Tango Overview

Using an intuitive example, we describe key insights and innovations Tango employs to satisfy the above requirements.

Example. Consider an enterprise network ASX in Fig. 3.1 that wants to *temporarily* offload real-time computing of user information to a small cloud in ASY. This cloud is particularly reliable and meets ASX's computing needs, but does not operate an edge close to ASX. Despite the existence of alternative low-latency paths from ASX to ASY, the BGP default path via AS1-AS2-AS3 incurs prohibitively high tail latency for ASX's real-time needs. ASX could benefit from using the cloud in ASY, if only it could forward its traffic via one of the alternative paths (*e.g.,* via AS5). Unfortunately, under BGP, ASX cannot use an alternative path, and relying on an SD-WAN raises concerns with privacy and cost.

Tango edges discover new paths and tunnel traffic over them. Tango exposes path diversity by treating IP prefixes as *routes* (as opposed to distinct destinations), and using unique prefixes to reach the same destination via distinct paths. To expose dis-

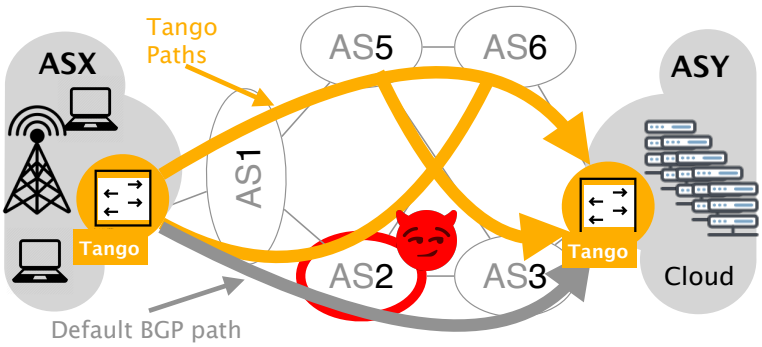


Figure 3.1: ASX (enterprise) and ASY (cloud) communicate over the BGP default (gray) path via AS2-AS3. Tango exploits collaboration between ASX and ASY to expose additional paths (orange).

tinct paths, Tango edges collaboratively use advanced BGP routing techniques available in today's networks (e.g., path poisoning and BGP communities). Note that Tango *automatically* constructs BGP advertisements to discover paths making Tango *plug-and-play* (§3.4). It also does not require the cooperation of any networks on the path other than the collaborative edges, and is thus *incentive compatible*.

As illustrated in Fig. 3.1, Tango exposes three distinct paths from ASX to ASY by advertising three distinct prefixes from ASY, which are already installed in core BGP routing tables, but unused by BGP. The Tango receiver (ASY) controls propagation of these advertisements through the Internet with BGP communities (§3.4). As a result, the Tango sender (ASX), can control which path it uses to send traffic to ASY by tunneling application traffic over the preferred path (§3.5.1). ASY decapsulates tunneled packets and forwards them toward their final destination. Even though the Tango sender can select a

different path for each packet, the BGP announcements made by the TANGO receiver are **stable** and each statically represent a distinct path. Thus, there are no BGP updates when the TANGO sender chooses to change paths (preventing BGP route flapping). In practice, TANGO is symmetric, and both edges can optimize bidirectional traffic.

TANGO EDGES COLLABORATIVELY MEASURE DELAY AND LOSS. Exposing path diversity is only the first step toward intelligent route control. ASX would need to monitor the performance of the four exposed paths to decide how to route traffic. TANGO provides highly accurate monitoring, as it operates at the edge of each network, avoiding access-network noise, *e.g.,* from wireless links. This gives TANGO an advantage over traditional end-host measurements that are notoriously inaccurate due to variable loss and delay within each network or probing techniques that can be deceived by ASes preferentially treating probes[20]. To passively measure delay and loss, TANGO adds the timestamp of when a packet left the sending edge network to every packet, along with a unique sequence number. Upon receipt, the receiving edge determines (*i*) *relative latency* between paths by calculating the difference between the time of packet receipt and the timestamp carried by the packet and comparing this with measurements from other paths; and (*ii*) loss by checking for missed segments (out-of-order TANGO sequence numbers).

Tango offers trustworthy loss and delay measurements. A rational (or malicious) AS, say AS2 in Fig. 3.1, might try to fool Tango into routing traffic through her infrastructure, not by improving the performance of her network but by compromising the monitoring or rerouting infrastructure. Since she cannot preferentially treat monitoring packets, as all packets are used for monitoring, she will try to fake lower delay by modifying the timestamps carried in the packets. Although adopting typical security primitives (*e.g.,* signatures, encryption) is challenging due to memory and computation constraints of modern high-speed hardware (*e.g.,* programmable switches), Tango protects both timestamps and sequence numbers of each packet from tampering. To do so, Tango leverage multiple insights. First, observe that timestamps and sequence numbers progress predictably. This allows Tango to precompute and prepopulate signatures with more flexible, memory-rich software. Additionally, observe that adversaries want to make their path look superior and thus have nothing to gain from replaying old signatures (*e.g.,* from old timestamps). This enables Tango to be resilient against replay attacks.

Tango supports fast and secure route updates. While one-way measurements are collected at the Tango receiver (ASY in Fig 3.1), the Tango sender (ASX) decides which path packets will take *e.g.,* based on per-class performance objectives. Instead of sending raw or summarized measurements back to the Tango sender (ASX), the receiver node ASY computes the best path for each traffic class according to ASX objectives and

freshly collected measurements. If the newly computed best path is different from the current one, ASY issues a separate route update to the sender in the data plane, broadcasting the update packet over all paths to the sender, for increased update reliability. To prevent an on-path adversary from tampering with the reroute updates, we use one-time-pads directly in the dataplane.

We stress that the need for secure data-plane measurements is not particular to Tango. In fact, many data-driven systems have been shown to be vulnerable to on-path adversaries [107]. Still, existing solutions focus solely on making monitoring scalable and accurate rather than secure [12,79,176,101], effectively overlooking trustworthiness.

3.4 Unveiling Path Diversity with PathFinder

Tango surpasses the limited path diversity offered with BGP by employing a novel recursive algorithm, PathFinder, which exposes BGP-compliant paths via *static* BGP announcements for different IP prefixes. Adhering to Tango's design requirements (§3.2.2), PathFinder does not assume collaboration of the Internet core or knowledge of the wide-area topology. PathFinder runs on two Tango edges with the minimum capability of announcing a single BGP prefix from one edge, and observing the AS path(s) for that prefix from the other edge. Increasing the test prefixes reduces the time it takes for PathFinder to expose all paths, but a typical duration using a single test prefix is $\approx$ 30

min. PathFinder will run at bootstrap of Tango to expose paths and rerun on rare occasions *e.g.,* if the Tango sender receives a BGP update regarding a route exposed by PathFinder. Rerunning Tango does not require Tango to go offline as long there is at least one unused BGP prefix.

PathFinder exposes previously unknown paths through the Internet by advertising prefixes to the Tango sender edge while *strategically blocking* (*suppressing*) *export of the BGP best-path*. PathFinder leverages two commonly supported route suppression methods: (*i*) BGP communities and (*ii*) BGP path poisoning. Community-based filtering involves attaching BGP communities supported by major transit providers (*e.g.,* those discussed in [26]) to suppress route exports (via no-export communities) or to lower route preferences so certain ASes do not use previously preferred routes. To support community-based filtering, PathFinder needs to be keyed with specific values of action communities supported by its upstreams and major transit providers. This can be obtained from publicly available routing guides. BGP path poisoning exploits BGP loop detection to prevent select ASes on the original path from importing the BGP announcements [24,122]. While these techniques have different topology-dependent trade-offs[1], they are largely interchangeable from PathFinder's perspective, both accomplishing the algorithmic objective of suppressing a given BGP route.

[1] BGP communities can suppress individual links but are not honored over provider-customer or peer-peer links while AS-path-poisoning only suppresses at the AS granularity but affects the entire path.

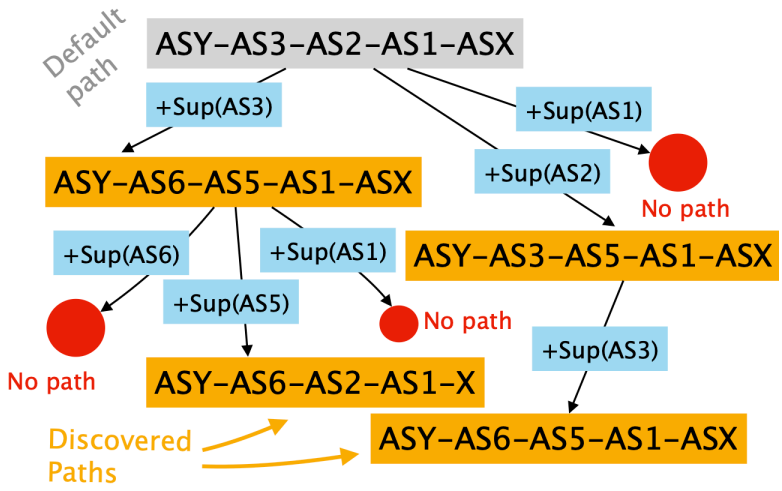


Figure 3.2: Finding paths via iterative advertisement suppression.

PathFinder recursively updates the BGP advertisements announced by one edge (the destination) based on real-time feedback from the other edge (sender). Specifically, PathFinder finds unidirectional paths between two nodes: a traffic source and destination. PathFinder starts by making a "default" BGP announcement from theTango destination. This announcement reaches the Tango source, which records the AS-path associated with this announcement. Note that without PathFinder, this would be the default and only BGP path available to the sender. Next, PathFinder *suppresses* the propagation of this route (using communities or path poisoning) to every AS on the recorded AS-path, effectively forcing the advertisement to find a different path to the source. For each path it finds, it recursively applies this algorithm.

As an illustration, to discover the paths in Fig. 3.1, ASX and ASY would need to jointly

construct a graph like that in Fig. 3.2. In this graph, nodes contain BGP paths between ASX and ASY. The root node represents the default path, while leaves capture paths discovered by PathFinder by supressing ASes in the edges of the path to the node. Red dots represent advertisement attempts that did not result in a path (*i.e.,* the advertisement did not reach the sender)[2]. Recall from Fig. 3.1 that the default path from ASX to ASY is via ASX-AS1-AS2-AS3-ASY (advertisements are propagated via ASY-AS3-AS2-AS1-ASX). To discover additional paths, ASY will propagate multiple routes starting from one that suppresses the propagation to AS3. By poisoning AS3, the advertisement will follow the path via ASY-AS6-AS5-AS1-ASX, and thus this will be the route that ASX hears. Next, ASY suppresses AS6 in addition to AS3 which results in no available path to ASX (ASX will hear no route), as all paths traverse either AS3 or AS6. Having fully investigated routes that suppress AS3, PathFinder backtracks to advertising a new route suppressing AS2 and then AS1.

The algorithm described above finds paths between senders and receivers that are distinct at an AS level. Thus, PathFinder does not leverage path diversity *within* each AS, or the existence of multiple peering locations for an AS pair. These paths could be found by combining PathFinder with Paris Traceroute [17]) and could be used by Tango by applying the source and destination port combos found by Paris Traceroute in the outer

[2] The graph is for illustrative purpose only and is constructed based on the information that PathFinder extracts (*i.e.,* is not part of its input).

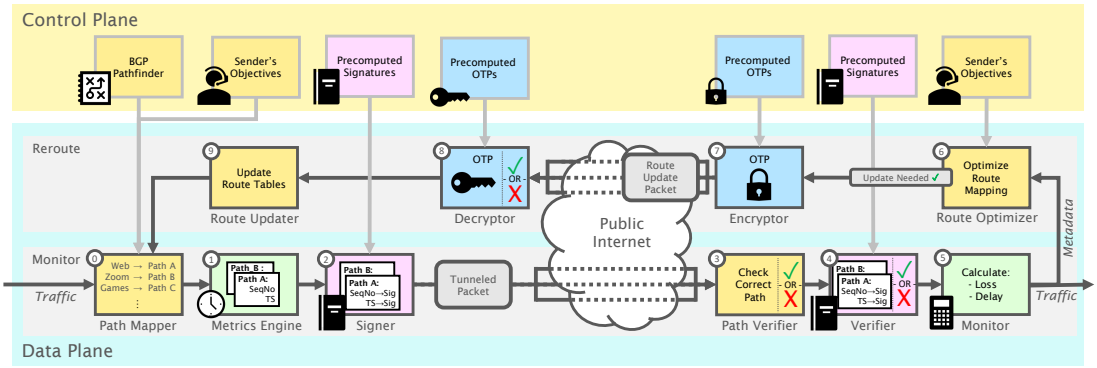


Figure 3.3: Tango is a hybrid control and data-plane approach to performant routing over the public Internet, informed by fine-grained, trustworthy telemetry. Tango relies on cooperation: both edge networks share secret keys and per-class performance objectives.

UDP header of the TANGO packets (the outer headers are ignored by the receiving switch). However, the use and exploration of additional intra-domain paths is out of scope.

3.5 SECURE, METRICS-INFORMED DYNAMIC ROUTING

With newly-exposed path diversity, TANGO can dynamically route traffic along paths best suited for given performance objectives, while being informed by fine-grained metrics. There are several challenges to accurate and trustworthy monitoring and rerouting. TANGO overcomes them with custom monitoring of one-way metrics (§3.5.1), trustworthy telemetry (§3.5.2), and real-time, tamper-proof route updates (§3.5.3). Fig. 3.3 illustrates the life cycle of a packet through each of these modules from a TANGO sender (left) to a TANGO receiver (right). We assume that TANGO routes traffic at the granularity of *traffic classes* to satisfy application-specific requirements[3].

[3] Further algorithmic details for TANGO's sender and receiver logic appear in the appendix of the published paper[27].

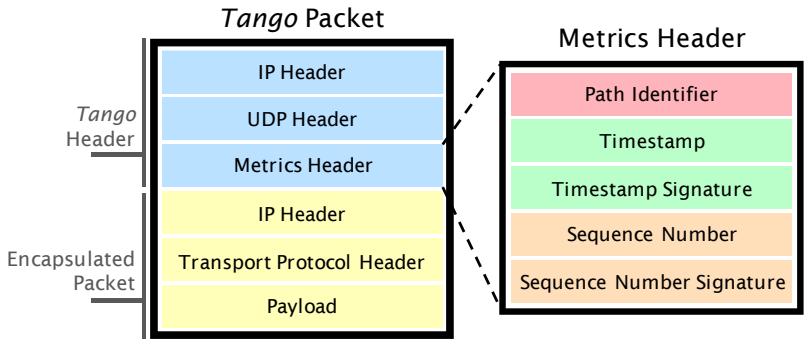


Figure 3.4: Tango tunnels application packets in an IP header specifying an interdomain path, a static UDP header to ensure consistent ECMP behavior, & a metrics header with signatures for integrity protection.

3.5.1 Multi-Path Monitoring

Tunneling for Multi-Path Routing ⓪. Tango tunnels packets through different physical paths by encapsulating them in a distinct per-path header (IPv4 or IPv6 headers) as shown in Fig. 3.4.[4] To route traffic according to its operator-designated traffic class objectives, Tango maintains a mapping from class to path identifier (PID), which uniquely designates a path, specified by an IP header.

Custom Per-Path Monitoring ①⑤ . In addition to the tunnel header, Tango adds custom header fields to enable per-path performance monitoring[92]. Unlike prior works that rely on TCP semantics to measure performance [12,133,42,111], Tango adds a cus-

[4]The Tango header adds several bytes to each packet, which could potentially cause MTU issues. Tango can resolve this in a similar way to other router-based encapsulation protocols by implementing Path MTU Discovery and responding with appropriate ICMP "fragmentation needed and DF set" packets, as specified in RFC 1191 [109].

tom *Metrics Header* which contains a 3-bit PID, 12-bit timestamp, 32-bit PID and timestamp signature, 8-bit TANGO sequence number, and 1-bit sequence number signature. The *Metrics Header* is lightweight (7 bytes overall) and used to identify packet routes, calculate loss and delay, and facilitate trustworthy telemetry.[5] The timestamps and sequence numbers are defined as follows:

1. *Timestamps:* The TANGO sender tags a packet with its local time (t_1) in ms before tunneling it to its peer. Upon receipt, the peer node records its local time (t_2) and calculates the per-packet delay as $t_2 - t_1$. The timestamp carried by packets is at the ms granularity. Since Internet path latencies are on the order of tens of ms, more fine-grained measurements would not yield increased benefits. Also, since TANGO measures relative latency across paths, clock skew between nodes is irrelevant.

2. *Sequence Numbers:* The TANGO sender also tags packets with a monotonically increasing sequence number (*s_curr*) before tunneling. The receiver tracks the highest sequence number seen (*s_seen*), calculating loss as *s_curr* − *s_seen*. As TANGO sequences are not re-transmitted if dropped (unlike TCP sequence numbers), each dropped packet is only counted once. Out-of-order packets increase loss but should be rare among packets with a fixed header traveling edge-to-edge.

[5] We developed these header sizes based on the parameters of the testbed we used. Even under more demanding production conditions, the anticipated header size will still be quite small compared to overall packet sizes.

Aggregated Monitoring ⑤⑥. Tango calculates per-path loss and delay metrics over an aggregation window of size i, by adding measurements to an aggregate sum. Upon the arrival of the i-th packet, the current sum will be persisted for later use in issuing reroutes, and the value will be reset. The operator may configure the window size, weighing tradeoffs in noise-resilience, network event response time, and security.

3.5.2 Secure Telemetry

Trustworthy Telemetry ②③④. To protect against tampering, the Tango sender cryptographically signs all timestamps, PIDs, and sequence numbers, ensuring integrity and authenticity. Upon receipt, the Tango receiver verifies each value to ensure they are untouched after transit over the public Internet. Signatures are path-specific, and hence secure against replay attacks: an on-path attacker cannot replace signatures with those they overhear from faster paths, as the signature is specific to the path (PID). Finally, any mapping (*i.e.,* pair of timestamps, sequence number, or PID to signature) that the adversary learns cannot be reused. The sequence number changes per packet. The timestamp changes every ms and older timestamps are not useful as the attacker only tries to make her path appear faster.

Practical Challenges for Scalability. Cryptographically protecting metrics at packet line rate is nontrivial. Cryptographic primitives are resource-intensive to implement

at scale. First, signing packets in the control plane requires laborious per-packet processing in software, while signing packets in the data plane would require multiple recirculations[41,172,175]—both too compute-intensive. Second, storing *precomputed* signatures directly on switch SRAM is too memory-intensive. Further, populating the data plane with precomputed signatures from the control plane creates read-write synchronization concerns between the data and control planes, as well as between Tango sender and receiver.

The Tango Signature Book ②④. To avoid online signature computation while still being memory-friendly, Tango creates a *signature book* of precomputed signatures, and periodically populates smaller precomputed *signature blocks* from the signature book to the data plane. The signature book is used for two operations: signing (on the Tango sender) and verifying (on the Tango receiver). The following insights help us deal with the memory challenge. First, we observe that both sequence numbers and timestamps are predictable (monotonically increasing) over time, enabling efficient pre-computation and storage. Second, we observe that to reduce delay the adversary would only need to guess one timestamp per ms, while to hide a single packet drop she would need to correctly guess many consecutive signatures. Further, sequence numbers will eventually wrap around, thus making old mappings irrelevant to the adversary. Hence, Tango uses 32-bit timestamp signatures and 1-bit sequence number signatures.

To deal with synchronization challenges, Tango employs two strategies. First, it uses

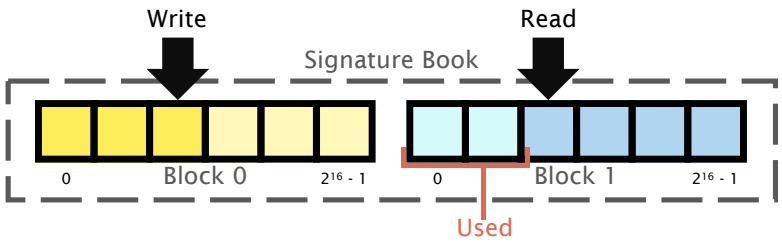


Figure 3.5: To deal with synchronization issues between reads and writes, Tango splits the signature book into two blocks, for reading and writing. The control plane writes fresh signatures to one block, while the data plane consumes the other for signing packets.

data-plane packets triggered by the control plane to quickly write signatures, instead of relying on control-plane write calls (*e.g.,* with gRPC). The control plane marshals precomputed signatures into large packets, which trigger block writes when processed in the data plane. Switch-specific constraints do not allow writing multiple indexes of the same register array, so signatures are sliced off packets in the data plane, which are recirculated until a null token is reached. This approach allows the control plane to update signatures faster than they are consumed by the data plane.

Second, Tango splits its signature book into two blocks, where at any given time, one block is being written and one is being read, avoiding a race condition between reads and writes (see Fig. 3.5). This approach makes the sequence-to-signature mappings time-dependent and thus deterministic. Critically, however, sequence numbers are not reset for every new interval. While resetting would be more economical from the memory perspective, it would also allow the attacker to silently drop all packets at the end of each interval. Instead, Tango

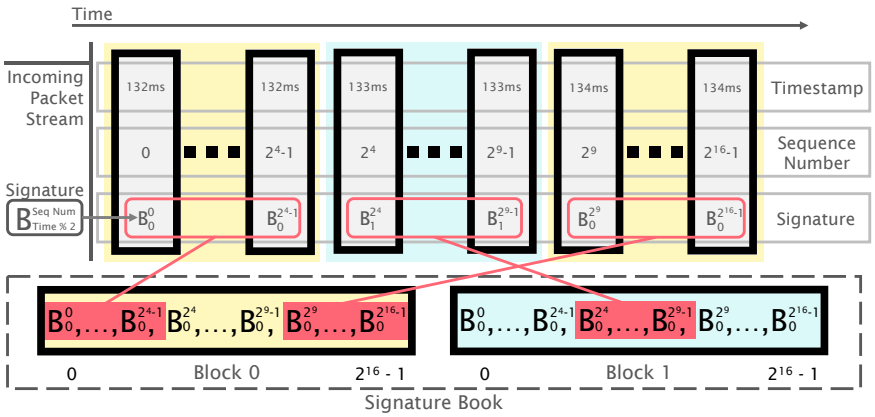


Figure 3.6: Tango ensures sequence numbers monotonically increase, even during block transitions, and only reset once all are expended. For example, assume block 0 is used on even timestamps and block 1 on odd. If, as the last sequence number of an even timestamp, the sequence number is $2^4 - 1$, the next sequence number, now with an odd timestamp and therefore accessing block 1, is 2^4.

stores the maximum signatures that can be consumed in each interval and ignores unused signatures. In other words, through every interval change, the sequence number continues to increase from its last value, as shown in Fig. 3.6.

3.5.3 On-Demand Reroutes in the Data Plane

Dynamic Route Updates ⑥⑨. The receiving Tango node issues route updates to the sender based on aggregated path metrics and the sender's objectives for each traffic class. As reroutes should be infrequent, Tango does not include these updates in the default Tango header, rather opting for a custom packet containing the route update (*i.e.*, the new PID for the corresponding traffic class).

Trustworthy Route Updates ⑦⑧. Naturally, a route update could be the target of an on-path adversary that (*i*) tampers with route updates to direct traffic to desired paths; (*ii*) injects route updates to move traffic from a path; or even (*iii*) drops route updates to cause a denial-of-service to the reroute mechanism. Observe that the first two attacks are catastrophic for Tango, since they have a direct effect on routing. Thus, to prevent tampering, the Tango receiver (which issues the updates) encrypts route updates before transmission, while the original sender decrypts and verifies the update before applying. To also protect against dropping of route updates, Tango broadcasts reroute packets over all available paths to ensure that at least one update will reach the sending node.[6]

OTP and Encoding Scheme. There are several challenges to encrypting updates: (*i*) limited compute and memory in the data-plane; and (*ii*) susceptibility of small ciphertexts to brute-force attacks. Tango solves these problems by using precomputed OTPs with extension encoding scheme. This is a natural solution as route updates are relatively infrequent yet unpredictable. However, since the number of possible reroutes increases with both the number of paths and the number of traffic classes, storing route-update signatures could be memory inefficient. Most importantly, compromised security of route updates would be catastrophic even for a single packet, necessitating the perfect secrecy of OTP.

Concretely, a route update consists of an index that selects the OTP to XOR with the

[6] Threat model assumes at least one path with no on-path adversary (§3.2.2).

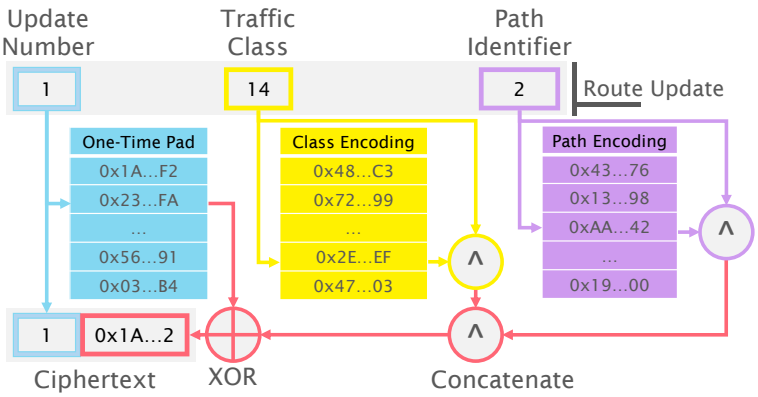


Figure 3.7: Integrity-protected route updates. The encodings, if sufficiently sparse, prevent adversaries from brute-forcing the few bits encoding class/path, since overall encodings can easily be verified.

concatenated traffic class and PID (see Fig. 3.7). The control plane periodically updates OTPs, such that they are only used once. While OTP offers perfect secrecy, it is vulnerable to bit flips, meaning the attacker can change a few random bits in the traversed route update to change TANGO behavior. To protect against this, TANGO independently encodes the traffic class and PID into 32-bit, sparse strings. As shown in Fig. 3.7, The value to encode (*e.g.,* traffic class) selects a static, sparse-bit string and appends it to form a single, 32-bit string. The concatenation of the encoded bit strings are XOR'd with the OTP, and sent to the peer TANGO node alongside the update number. If an adversary tampers with the update number, traffic class bits, or the PID bits, the decrypted update or the encodings will be incorrect, and the update can be safely ignored. If the adversary were to now try and brute-force the update, they must now brute-force all encoding bits, which are tied to encoded values.

Figure 3.8: Global Tango testbed across 25 geographically distributed Vultr data centers.

3.6 Internet-Scale Measurements

We showcase Tango in the wild by deploying it with eBPF [152] (§3.7) on 25 nodes in globally-distributed data centers of the cloud provider Vultr [162] (see Fig. 3.8). Through our Internet-scale measurement study (§3.6.1), we prove that Tango uncovers *path diversity* even in a single-homed environment, where conventional BGP only offers a single path. We also show that Tango uncovers *performance diversity* in these alternate paths, which often outperform the default BGP path by up to 100% of the time with up to 39% lower latency. To further validate and generalize these results, we also perform an Internet-scale simulation that confirms the rich path diversity available in the public Internet (§3.6.2). Together, we showcase the benefit, practicality, and incremental deployability of Tango.

Ethics Statement: We note that all testbed infrastructure in edge networks of our Internet-

scale study are operated by the authors, and that traffic sent from Tango nodes across the public Internet are transparent to intermediate ASes and can be processed as normal application traffic at reasonable sending rates (up to 1Mbps), raising no ethical issues.

3.6.1 Operational Deployment

Choice of Edge Network. We perform our measurement study with Vultr as a deployment convenience, however, many other types of edge networks can easily benefit from Tango. Characteristics that would make an edge network particularly amenable to Tango include having available prefix space (either IPv4 or IPv6) and either BGP communities *or* path poisoning capabilities. Note that the edge network does not require a provider that actively *engages* in BGP community announcements, as long as they are able to *transit* communities. This is often the default behavior in ISPs [146,147].

Methodology. To measure available path diversity, we ran PathFinder on all bidirectional paths between 24 of the Vultr data centers (552 pairs in total)[7]. Vultr allows customers to make BGP announcements, supports several action communities (to control its own BGP behavior), and transits BGP communities (to potentially impact the behavior at remote ASes) [160,161], but does not allow customers to do path-poisoning. Thus, we ran PathFinder using community-based suppression, inputting BGP suppression com-

[7] During PathFinder exploration, only 24 of 25 nodes were available.

munities supported by Vultr as well as by several major transit providers (specifically ASes 3257, 6453, 4755, 3356, 1299, and 174).[8] While Vultr does export a full BGP route table to customers participating in its BGP services, it only provides a single next-hop at each datacenter: the Vultr upstream router. Thus, *by default, each source-destination pair would only have a single path, without* TANGO.

We also used TANGO to measure performance diversity between 23 Vultr pairs, with Vultr Stockholm fixed as the receiver. Specifically, we generated 1Mbps `iperf` UDP flows across 7 different TANGO-exposed interdomain paths for all 23 pairs, passively measuring latency and loss at 10 ms measurement intervals over a period of roughly 32 hours. We also repeated measurements with an additional 23 node pairs, using Vultr LA as the fixed receiver, and observed similar results; see the appendix of the published paper[27]. We present results for Stockholm measurements below.

PATH DIVERSITY OVER THE PUBLIC INTERNET

HOW MUCH PATH DIVERSITY CAN TANGO EXPOSE? Of the 552 node pairs explored, PATHFINDER exposed alternative paths for *503 pairs* (*91%*), *unveiling opportunities to benefit from multi-path routing*. The CDF of number of paths exposed per Vultr pair is included in Fig. 3.9. The median number of paths available between two nodes was 3, and

[8] As there are no standard values for many BGP route suppression communities[26] these values had to be found by hand from routing guidelines.

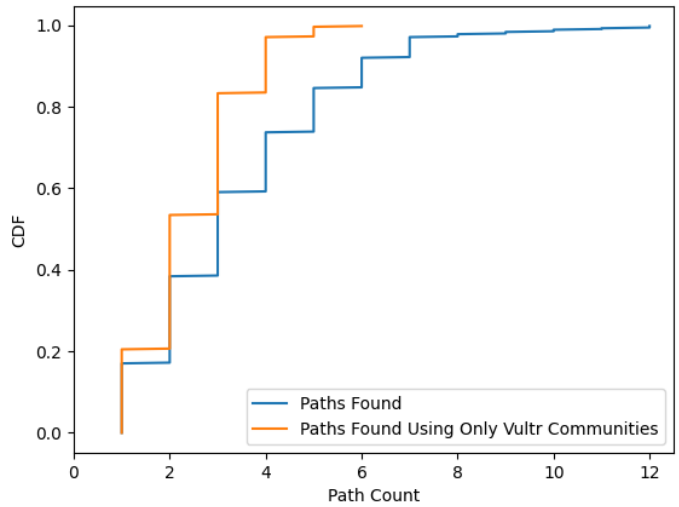


Figure 3.9: Available paths identified with PathFinder, per Vultr pair using only communities supported by Vultr and communities supported by Vultr as well as ASes further down the path.

the average was 3.3. PathFinder also uncovered a sizeable long tail, as 84 different pairs had 6 or more paths and 3 pairs had 12 different paths between them. In some cases our BGP announcements appeared to be filtered as the source node used did not hear the BGP announcement made from the destination node. These cases are recorded in the CDF as having a single available path because Tango can still function by using the Vultr-provided IP address of instances to tunnel traffic.

We also explored how many of these paths were found using only communities supported by our immediate transit provider Vultr and how many were found because of BGP community support at ASes further down the path (shown as separate traces in Fig. 3.9). If only Vultr-supported communities are used, the median pair of nodes only has two paths,

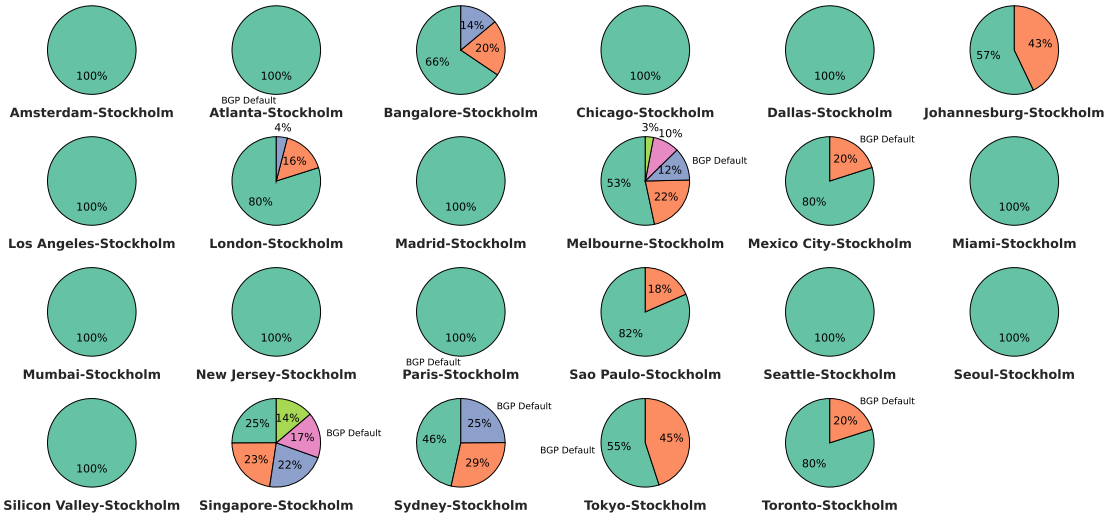


Figure 3.10: The BGP default path is beaten by one or more Tango-exposed paths for 20 out of 23 node pairs, for 4 out of 23 pairs it is beaten by 3 or more paths, while for 2 out of 23 pairs it is beaten by 4 alternative paths.

but using transitive communities raises this by 50% to 3 paths. The maximum number of paths found between any two pairs of nodes increases from 6 to 12 showing the importance of community support at various nodes in the topology.

The average run time for PathFinder was quite low (<1h) as each pair only required 6.8 BGP announcements on average (with 5-min separations to account for propagation). While already encouraging, the number of paths can be significantly increased if Vultr were to allow BGP path-poisoning. Unlike community-based suppression which is only supported by some ASes and varies in implementation, AS-path poisoning is mandated by BGP RFC [126], though some networks filter announcements with certain ASes poisoned [144].

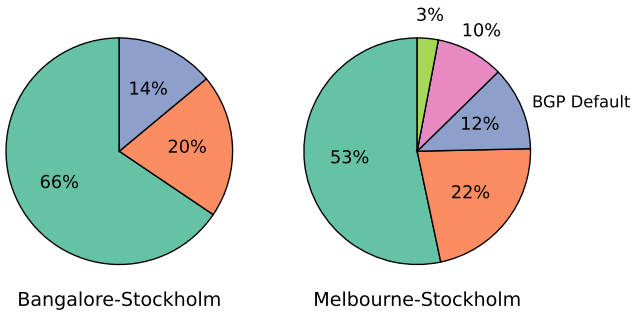


Figure 3.11: Breakdown over 32 hours of how often a path had lowest relative latency. Here, the BGP default is beaten by alternative paths 100% of the time (Bangalore) and 88% of the time (Melbourne).

Performance Diversity across Exposed Paths

How often are Tango-discovered paths better than the default? We define the best path to be the one with lower relative one-way-delay compared to all other paths. Of the 23 pairs in our Vultr Stockholm measurements, *20 pairs had an alternative path that outperformed the default BGP path for a significant amount of time*: 100% of the time for 15 pairs, and 75-88% of the time for 5 pairs (Fig. 3.10). This clearly shows the benefit of deploying Tango. Of the remaining three pairs (Atlanta-Stockholm, Paris-Stockholm, Tokyo-Stockholm), the default path was dominantly best for only two pairs and was best only 55% of the time for the last pair. We also note that there were many pairs with *more than one* alternative path which outperformed the default, providing further path diversity and potential performance resilience: 7 pairs had two or more alternatives and 4 pairs had three or more alternatives. In Figs. 3.11 and 3.12, we visualize results for two node pairs, showing alternative paths outperforming the default.

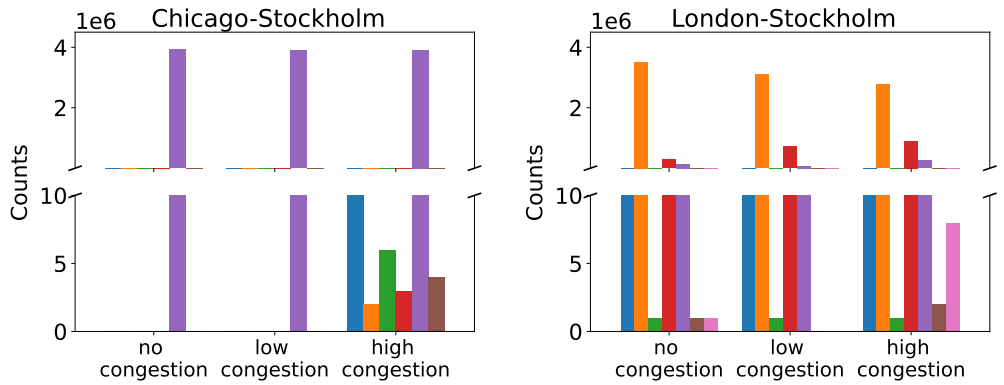


Figure 3.12: Number of times different paths emerged as the best for a given congestion condition, per node pair. "no", "low", and "high" congestion conditions correspond to the lowest, middle, and upper tertiles, respectively, of one-way-delay experienced by the best path. While one path might dominantly be best under no/low congestion, during high congestion more paths emerged as the best path.

How long is one path the best? We measured the window of time one path remained the best for each of the 23 Vultr pairs (Fig. 3.13). Path longevity varied across paths, likely due to performance variations from changing network conditions (*e.g.*, congestion). For 9 pairs, the median window duration was 350-925*ms* showing that dynamically updating the used path would be beneficial for forwarded traffic. For 11 pairs, the median duration was shorter at 10-30*ms*, most likely due to ECMP behavior between the nodes. The average duration for these 11 pairs ranged between 95-99*s* (from New Jersey, Mumbai), 121-221*s* (from Toronto, Tokyo), and 105-107*s* for the 7 other pairs. For the last 3 pairs, the median was much longer, lasting for 23-105*s*, while the average duration for each pair was 107s (from Johannesburg, Madrid, Chicago).

Tango performs dynamic reroutes when a new best path emerges for longer than a

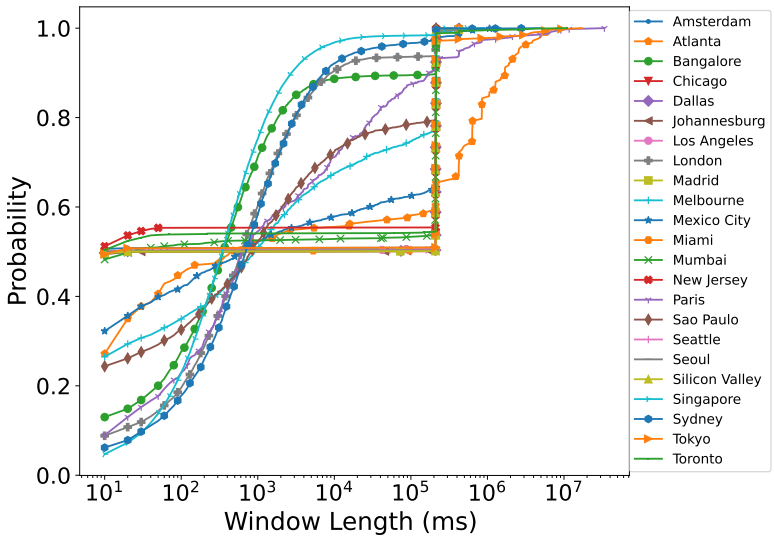


Figure 3.13: On average across all 23 pairs, best paths lasted from 5-541s, with median durations of 10ms-105s. Tango only chooses to perform dynamic rerouting for windows longer than 100ms.

given window threshold. The exact threshold can be configured per pair using such measurements. For instance, with this threshold, 10-30ms windows would not trigger a route update. Network operators can increase this threshold to provide more static long-term path optimizations, or further lower it for even faster dynamic updates.

BY HOW MUCH DOES THE BEST PATH BEAT THE BGP DEFAULT? We measured the latency difference between the best and default paths, for each of the 23 Vultr pairs (Fig. 3.14). We found that for five pairs, the best path had at least 20% lower latency than the default for more than 6 hours within the 32-hour experiment. To better grasp the expected benefit of TANGO, consider that these nodes are in different sites of the same cloud. While as noted previously, there were two pairs for which the default path was best (Atlanta-

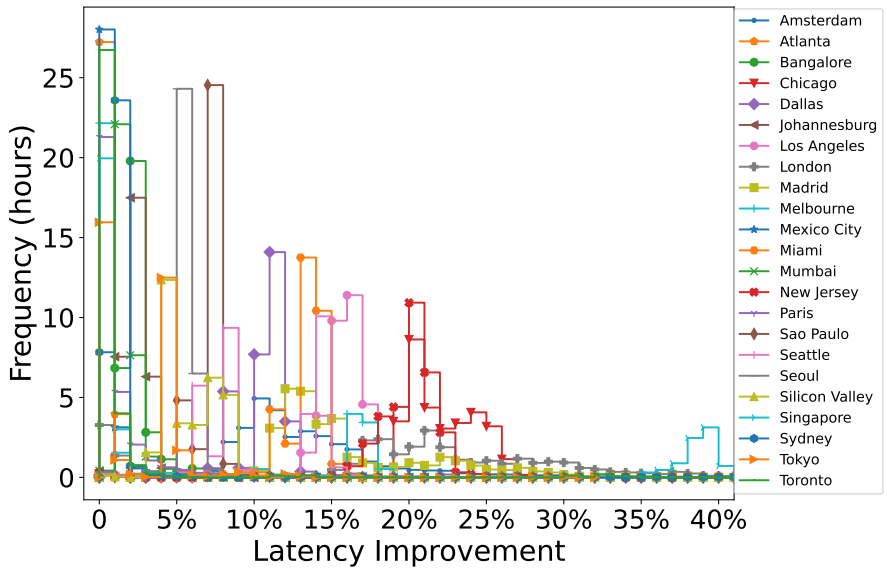


Figure 3.14: Best paths that Tango exposed outperform the BGP default by up to 22% on average, and up to 39% for some pairs.

Stockholm and Paris-Stockholm), for the remaining 21 pairs, *the non-default best path outperformed the default by an average of 1-22%* for durations of 0.9-24.54 *hours*. Further, the 75th percentile delay improvement was 24% for Chicago-Stockholm and 26% for London-Stockholm, lasting for 4.07h and 0.85h, respectively.

Event Analysis

While Fig. 3.14 demonstrates the benefit of running Tango for the average-case, another major advantage of Tango is to rapidly avoid problematic performance events by moving to other paths during disruptions. To quantify this benefit, we searched our measurements for significant performance degradation *events* by filtering for 10 consecutive measurements

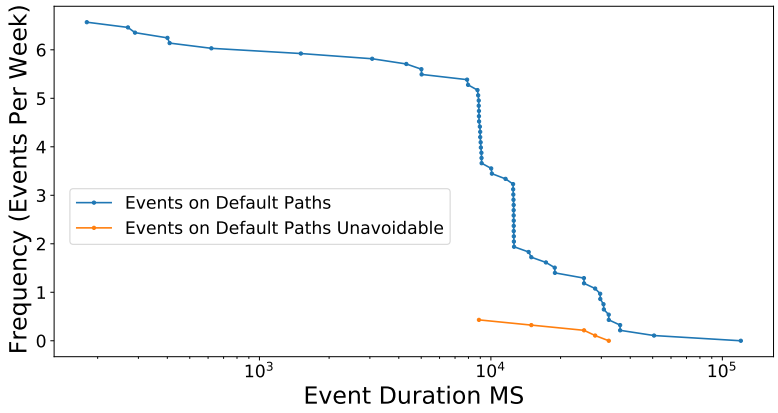


Figure 3.15: Occurrence frequency vs. duration of high-loss performance degradation events on default BGP paths. Tango could protect edges from four high-loss events on average per week.

(i.e., a 100ms window, with data points taken every 10ms apart) which were 20ms higher than the baseline one-way-delay on that path. We determined an event to be over once 10 consecutive measurements were below our event threshold. For each event, we computed the relative average one-way-delay and loss during the event. We also marked each event as avoidable if there was another path between the same pair of nodes that was not experiencing a performance degradation event. We computed the frequency of events on paths by dividing the total number of events by the total duration of measurement collection.

From there, we further selected events with more significant performance degradation in loss or delay, potentially compromising real-time applications. We do expect that such events would cause customers to complain even if the average network performance is good. Concretely, we selected events with greater than 20% loss or 100% one-way-delay increase. Fig. 3.15 presents a graph of event duration vs. frequency for loss events on de-

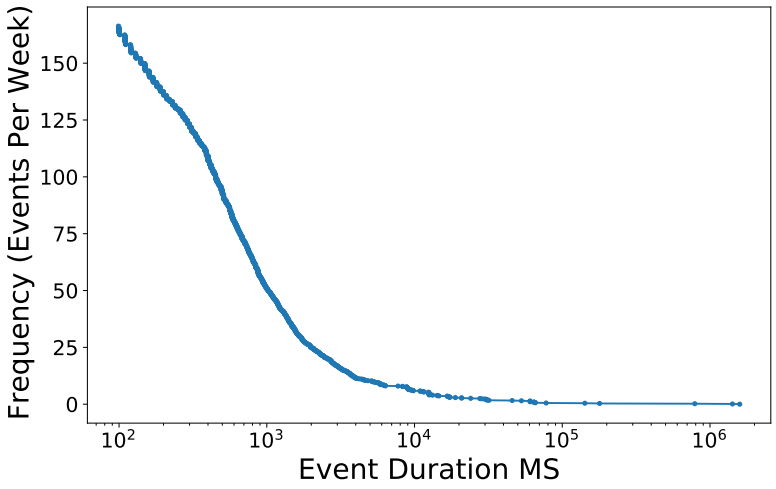


Figure 3.16: Occurrence frequency vs. duration of high-latency performance degradation events on default BGP paths. All events would be avoidable with Tango.

fault BGP paths (Tango-discovered paths experienced similar events). In the absence of Tango, events with 20% loss for longer than 8 seconds occurred over 5 times a week. Meanwhile, with Tango-exposed paths and adaptive routing, such loss events can be reduced to *less than once a week on average*. It should be noted that in our dataset, all unavoidable events occurred at a single sending node: Vultr Brazil (which may have seen a higher number of correlated events due to potentially less path diversity in developing regions like Brazil). *For all other nodes, every high-loss event encountered could have been avoided with Tango.*

In addition to looking for events with high loss we searched for events with high latency (i.e., a 100% latency increase over the baseline). We found these happened more frequently than high loss events. Fig. 3.16 shows these events. For high latency events, we found *all*

events were avoidable with Tango.

3.6.2 Internet-Wide Simulation

To understand potential path diversity across the Internet topology in an even more generalized setting, we counted Gao-Rexford-compliant[66] paths between randomly-chosen ASes in the March 2020 CAIDA Internet Topology Dataset[37]. To optimize the counting process, we separated a Gao-Rexford path into two sections: an initial part consisting of zero or more customer-provider links joined (optionally by a peer-peer link) to zero or more provider-customer links. With this in mind, for each BGP destination considered, we divided the topology into two sections: a "provider cone" that was reachable only traversing customer-provider links, and all other ASes outside the provider cone. We counted paths to all ASes in the provider cone via only customer-provider links by treating this as a Directed Acyclic Graph (DAG) based on the Gao-Rexford premise that there are no customer-provider loops[66]. We similarly counted paths from all other ASes to the source AS by treating the region outside the provider cone as a DAG but traversing provider-customer links (instead of customer-provider links). This gave us a path count to all ASes in the topology using either customer-provider or provider-customer links depending on whether the ASes were inside or outside the provider cone. Finally, we joined these two counts to form full paths by traversing each AS A in the provider cone and counting the potential paths that contained A in the last customer-provider link (the sum of the paths through ASes not

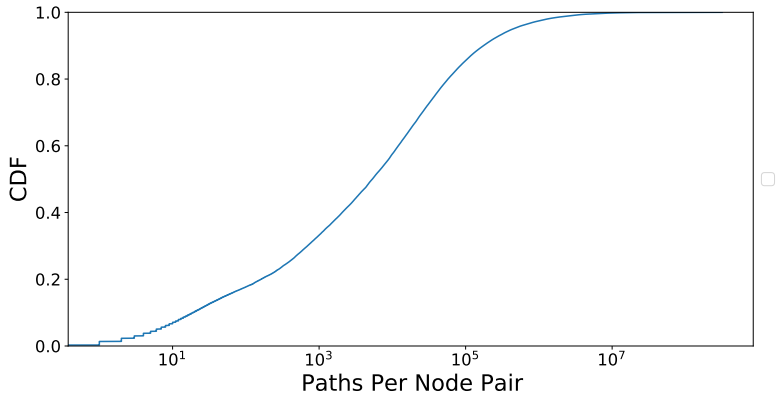


Figure 3.17: The number of Gao-Rexford-compliant paths between 999k random topology node pairs. PathFinder exposed more than 2 unused BGP-compliant paths for 98.6% of pairs.

in the provider cone of the peers and customers of A multiplied by the number of paths through the provider cone to reach A). This algorithm is scalabe and ensures counting distinctness, it is actually a lower bound on the number of paths in a given topology, as ASes in the provider cone can potentially be reached over peer-peer or provider-customer paths as well, which our counting methodology does not permit. Our counting is also limited by the accuracy of the CAIDA AS-Relationship dataset as is standard with many other Internet-scale simulation work[26,24,25,150].

We ran our counting algorithm to count Gao-Rexford-compliant paths between 1000 randomly-chosen ASes producing 999,000 distinct source-destination pairs. We found today's Internet topology offers rich path diversity, as the median AS pair had 5,323 Gao-Rexford-compliant paths (Fig. 3.17). We largely attribute this richness to the high degree of certain vertices, including large IXPs, on the Internet graph; indeed, in a richly-connected

topology paths grow exponentially. We do not advocate for Tango to use more than a couple of them, yet this result shows potential for multi-path routing across the wide area.

3.7 Internet-Scale Route Control

We showcase Tango's real-time route control with an end-to-end experiment spanning North America and Europe, illustrating how Tango leverages collected measurements to perform dynamic route control and avoid performance degradation events (§3.7.1). We also present microbenchmarks that highlight the operational feasibility and efficiency of performing line-rate integrity protection with Tango's switch prototype (§3.7.2).

Implementation. We implemented Tango's data plane logic on modern switch hardware and with eBPF on standard Linux servers. Our switch prototype was implemented in 199 lines of Lucid code [145] and compiled to 1279 lines of P4 [31] targeted for an Intel Tofino programmable switch [81]. Our interchangeable eBPF version was written in 401 lines of code. Meanwhile, Tango's control plane component and eBPF loading program was written in 524 lines. We have released Tango's source code on GitHub[9].

Testbed. Our hardware testbed consisted of an Intel Tofino Wedge32X-BF programmable switch and several servers, each with a 20-core Intel Xeon Silver 4114 CPU and a Mellanox ConnectX-5 2×100Gbps NIC. Our sending edge network was based on Princeton Uni-

[9] https://github.com/PrincetonUniversity/tango-routing

versity's campus in North America, with a server (running Ubuntu 20.04 and kernel version 5.4.0) sending main application traffic and generating background traffic flows, and the switch running Tango data plane logic. Meanwhile, Tango's receiving edge was deployed with eBPF on a standard Linux server at a Vultr data center in Stockholm, Sweden running Ubuntu 22.10 with kernel v5.19.0. The eBPF programs were built with Libbpf and Clang 15.0.6.

3.7.1 Dynamic Reroutes

To evaluate Tango's end-to-end dynamic route control, we announced seven distinct IP prefixes from Vultr Stockholm via BGP, using community sets that we previously found were optimal for finding paths between Stockholm and other locations (§3.6.1). We did not have access to a BGP feed from our institution so we could not rerun our PathFinder algorithm specifically for this pair of nodes. We also noticed ECMP being used for outbound traffic from our institution. To benefit from this we explored the round-trip-time when sending traffic to different destination IPs within the same BGP prefix. We found two available ECMP paths for every announced BGP path. After enumerating these 14 paths, we observed some paths had seemingly-identical performance[10]. Ignoring redundant paths we had 12 distinct paths.

[10] This is because we could not see the BGP path used by our institution so some BGP community combinations produced identical paths.

We generated keep-alive flows along all exposed paths while continually collecting per-path metrics. We also had one active flow, which we monitored for potential benefits from dynamic routing. We developed a simple control algorithm based on relative one-way-delay that would move the active flow to a different path if that path outperformed the current path for 10 consecutive measurements (i.e., 100ms for measurements taken every 10ms). We found this algorithm to be stable across the wide area, only rerouting during significant network degradation events. Once the control algorithm determined a route change was necessary, it produced a reroute data packet which was sent back to the Tango switch node at our institution. This caused a register update on the switch, which rerouted the active flow to a different path.

To observe Tango's potential to dynamically avoid events like those discussed in § 3.6.1, we utilized another Intel Tofino switch running between our Tango switch and the Internet to inject delay (with packet recirculations) on specific paths. To make our experiment more realistic, we replayed delay events that we had observed in the wild on paths that (*1*) had a similar one-way-delay as that of our institution to Stockholm and (*2*) had the largest number of disruptive events.

When we ran this experiment, our control algorithm started by moving the active flow from the BGP default path (path 0) to the optimally performing path (path 5). As the experiment continued, our delay mechanism began simulating a performance degradation

[10] It is likely a congestion-induced single-packet spike around 1500 ms.

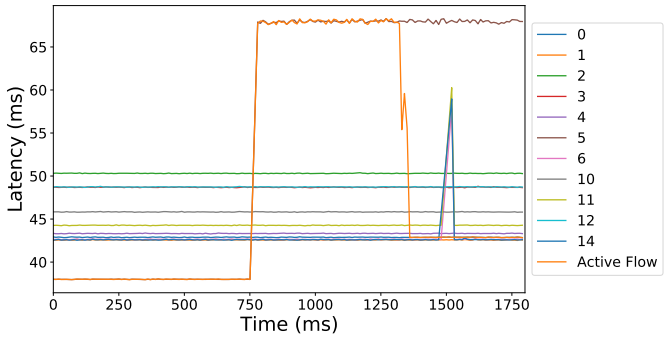


Figure 3.18: One-way-delays of different paths and an active flow during our simulated performance degradation event.

event, causing path 5 to have its one-way-delay spike. *Within 650ms after the first sign of performance degradation, the observed active flow was moved back to path 0* (see Fig. 3.18). Even with a simple control loop, Tango can reduce the duration of 90% of the high-loss events observed in §3.6.1, offering more than a 10-fold reduction in the median length (12s) of high-loss events. This response time could potentially be even faster if our prototype eBPF module generated reroute packets in the kernel as opposed to utilizing a separate user-space process (written in Python and taking approximately 400ms to initialize and execute), or if a more aggressive control algorithm was used.[11]

3.7.2 Data Plane Microbenchmarks

To demonstrate the operational efficiency of our integrity-protection scheme, we analyzed theoretical sizes for Tango's signature book (§3.5.2), experimentally evaluating required

[11] Our test flows ran UDP so reordering was not a prominent concern. If this rerouting algorithm is used on TCP traffic, flowlet boundaries can be taken into account to minimize overhead due to packet ordering changes.

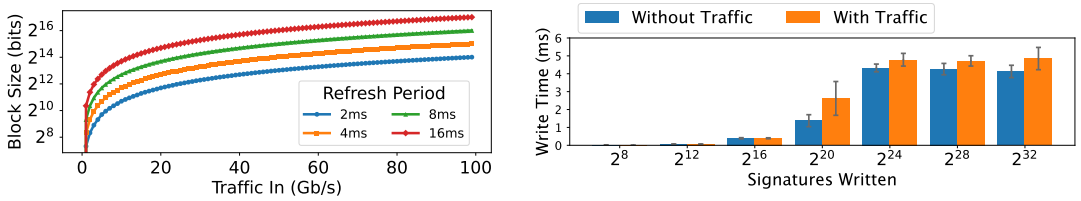


Figure 3.19: Required signature block size (*left*) and write speed (*right*) reveal the practicality of a precomputed data plane signature scheme.

control plane write speeds for book population. Our results prove that, for reasonable packet sizes, required block sizes refreshed every 5–15ms can comfortably be stored in the data plane and populated from the control plane (Fig. 3.19).

Book Block Hybrid Write Speed. We experimentally confirmed that the control plane can write an entire signature block across all books at speeds well below the refresh period, ensuring all signatures are refreshed before the data plane moves to the next block. Our first server issued control plane writes, the second generated background traffic at 100Gbps with DPDK[61], and the switch ran our slice-and-recirculate write behavior (§3.5.2) while tunneling out background traffic. We gathered write-time measurements by taking the time difference between the arrival of the first and last signature written to the block. As shown in Fig. 3.19, at a refresh period of 8ms at maximum port speed, the control plane can write 2^{16} signatures in less than .6ms over the data channel. Meanwhile, the required 2^{20} signatures for all blocks across all books can be written in approximately 1.3ms without background traffic and 2.6ms with it, while being well below the 8ms refresh limit.

3.8 Related Work

Performance-Driven Routing. Many prior works have explored performance routing; however, they suffer from poor deployability. Traffic engineering works such as TeXCP[88], Edge Fabric[129], and Espresso[171] assume that the edge network is multi-homed. While Edge Fabric and Espresso show the promise of SDN-based performance-driven routing, they can select between multiple already-existing BGP routes, thanks to global PoPs and peerings from Meta and Google. Such infrastructure is infeasible for smaller network operators and cloud providers. Other data-driven routing solutions such as Blink[79] and ShortCut[138] allow fast failover in the data plane but do not improve performance. RouteScout[12] provides metric-driven dynamic routing in the data plane, but struggles to provide accurate metrics and also assumes multi-homed networks. AnyOpt[179] optimizes anycast catchment but does not advertise multiple prefixes for the same destination. PECAN[158] does advertise different routes to the same destination but only steers between them with DNS, preventing the fine-grained route control Tango offers.

PAINTER[95] similarly uses multiple IP prefixes to advertise different routes for inbound traffic, but is designed for a cloud environment, which significantly changes the design space. First, as PAINTER runs in a highly-peered cloud, it can simply advertise distinct prefixes to different immediate neighbors, and does not attempt to find distinct paths after the initial hop (making it useless to single-homed networks). Second, PAINTER does not

perform data-plane telemetry and leverages application-layer telemetry enabled by proxies running in edge networks. Tango innovates on data-plane telemetry and does not require application proxies, while being protocol-agnostic and robust against malicious intermediate networks. Finally, Tango is designed to be deployed in programmable data planes like P4 switches and smartNICs. PAINTER utilizes proxies running on traditional CPUs that do not have the scaling and cost benefits of data-plane hardware.

Secure Telemetry. Multiple solutions for real-time data-plane telemetry exist [124,133,99,112], yet they are not designed with security in mind *i.e.,* their results could be compromised by an adversary. Moreover, they often require collaboration of all switches/routers in the path. For instance, INT (and later versions) [92,139] collect fine-grained performance metrics at each hop, enabling informative network monitoring to operators, but require each switch to implement the protocol, which is not a reasonable assumption in the wide-area setting. A few secure telemetry solutions exist, such as Stealth Probing [20] and path-quality monitoring [67], but they do not provide the necessary fine-grained metrics for real-time, dynamic routing, and are not implementable in today's hardware.

Data-Plane Encyption. There are several general-purpose encryption schemes ported to the data plane; however, they are too resource-intensive. For example, the Advanced Encryption Standard *i.e.,* the de facto cipher for most Internet applications AES-Tofino [41],

utilizes the majority of Tofino memory resources and would need, optimally, 5 pipeline passes for each 16-byte block to encrypt. Even more lightweight cipher deployable on an ASIC such as Simon and Speck[22], ChaCha[175], and HalfSipHash[172] are still too resource intensive. Beyond memory, they require several recirculations for every plaintext block to be encrypted. Other solutions, such as Raven[165] and Pinot[164], are application-specific and not easily extensible to other use-cases.

3.9 Conclusion

Tango is the first route-control scheme to expose multiple wide-area paths without the cooperation of the Internet core, while offering accurate and trustworthy edge-to-edge measurements. Our Internet-wide experiments show there are significant benefits from optimizing routing on the public Internet, using Tango-exposed paths not available with BGP. We show Tango can run on a hardware switch or with eBPF, making it practical even for small networks.

4

Optimizing Traffic Allocation Across Paths for Privacy and Performance

In this chapter, we present PraxiGuard, which complements the previous chapter by determining how newly exposed interdomain paths should be used to advance *route control*. Whereas Tango showed how cooperating edge networks can expose additional wide-area

paths beyond the default route, PraxiGuard shows how traffic should be allocated across those paths to protect user privacy—specifically, against website fingerprinting (WFP) attacks—while preserving performance. We show that, through selective cooperation among edge networks, PraxiGuard can systematically model the privacy and performance properties of available paths and automatically compute optimal traffic splits subject to the network's operational capabilities, without relying on overlays, protocol-specific workarounds, or end-host modifications. Our evaluations on real-world path substrates demonstrate that PraxiGuard reduces worst-case fingerprinting risk by a median of 52 percentage points (up to 99 pp), while preserving application performance (*i.e.,* browser page load times). Moreover, PraxiGuard provides additional privacy gains over a uniform splitting baseline for 56% of evaluated Tango cases, with gains of up to 77 percentage points.

4.1 Introduction

Online privacy is under increasing pressure from state-level surveillance and network-based censorship. Governments worldwide routinely monitor Internet traffic and restrict access to information, undermining free expression and user autonomy [102,127,50,29,63,62]. While privacy-enhancing technologies—such as virtual private networks (VPNs) (e.g., WireGuard), anonymity networks (e.g., Tor), proxy-based systems (e.g., Privoxy), and privacy-focused browsers (e.g., Brave)—hide the identities of endpoints, they leave traffic vulnera-

ble to traffic-analysis attacks[156,118]. In particular, *website fingerprinting* (WFP) attacks exploit observable traffic patterns such as packet sizes, directions, and timing to infer a user's browsing activity, even when traffic is encrypted or routed through privacy-enhancing systems. Prior work has demonstrated that WFP achieves high accuracy under various threat models, posing serious risks for users in censored or surveilled environments[118,168,140].

To thwart a WFP attack, defenses must obfuscate the traffic fingerprint in some way. Existing solutions fall into two broad categories: additive obfuscation and subtractive obfuscation. Traditional additive techniques—padding packets, injecting cover traffic, introducing inter-packet delays[34,85,71]—alter the trace by *adding* noise. This approach can reduce fingerprintability, but inherently imposes overheads in bandwidth, latency, or both. For users seeking privacy during ordinary browsing, this performance degradation may be an unacceptably high cost. Alternatively, rather than adding cover traffic or artificial delays, recent *subtractive* approaches[53,78,167] attempt to disrupt observable signals by splitting traffic across multiple connections, interfaces, or overlay paths. While traffic splitting is a compelling direction for low-overhead protection, existing approaches fail to realize this promise for *ordinary* web traffic, or against Autonomous System (AS)-level adversaries that observe traffic over network paths. These works fall short for two fundamental reasons.

First, existing splitting defenses are largely oblivious to the underlying Internet topology that the paths traverse. Prior works often assume that separating traffic across distinct con-

nections, interfaces, or overlay paths is sufficient to improve privacy[53,78,167]. At the AS level, this assumption does not hold: even with distinct first hops, traffic that appears separated at the endpoint can still converge at common downstream ASes, allowing an observer to see a substantial fraction of the original trace. This vulnerability is exacerbated by the Internet's growing centralization, where a small number of transit ASes carry a disproportionate share of global traffic[151]. Moreover, while mechanisms like client multihoming may expose multiple access links[78,167], they neither reveal nor control the downstream paths traversed. Thus, without topology awareness, naïve traffic splitting leaves users vulnerable to the network's most powerful observers.

Second, existing designs are architecturally misaligned with the privacy needs of the general public. Anonymity networks like Tor provide vital protection for a niche set of ultra privacy-conscious users, but they are built for a threat model of total distrust where the user must be hidden from even their own edge network. This threat model necessitates complex multi-hop routing and software-based processing that incur significant performance penalties[7,14,143]. While a necessary trade-off for full anonymity, this overhead is prohibitively burdensome for users who simply seek to protect their everyday browsing traffic from on-path observers, while being unable or unwilling to sacrifice their user experience.

PraxiGuard addresses this gap by leveraging the existence of edge networks that are trusted by, and structurally incentive-aligned with, the users whom they serve. Unlike

anonymity models designed for total distrust, our work focuses on environments—such as campus networks, enterprises, or ISPs—where users already rely on their edge provider as a natural point of aggregation. In these settings, traffic typically traverses shared gateways, NATs, or proxies; consequently, external observers already associate activity with the edge network rather than a specific internal host. Here, our goal is to provide edge-assisted privacy for everyday users by empowering trusted edges to act on their behalf and systematically mitigate risks posed by AS-level adversaries.

This approach is particularly compelling for edge networks that prioritize privacy as a core service or competitive advantage. For example, consider a journalism non-profit whose staff frequently access sensitive services, or a corporation seeking to protect trade secrets—where observing traffic patterns to specific external sites could reveal a company's strategic direction or proprietary research. Alternatively, an ISP might want to offer built-in privacy with good performance as a competitive advantage to attract a broader user base. With PraxiGuard, these entities can leverage their network vantage points to minimize AS-level exposure, protecting ordinary application traffic without requiring technical expertise from the end-user.

For such privacy-motivated edges to mitigate AS-level risks, they must first have access to more than the single path provided to them by default Internet routing. PraxiGuard obtains this diversity through edge-accessible path-control substrates: Tango[28], which un-

covers existing alternative interdomain paths on today's Internet, and SCION[121], which provides explicit path choice through SCION-IP gateways (SIGs). Our analysis reveals that both substrates expose significant AS-level diversity across a wide range of source–destination AS pairs. Relative to the reference path,[1] 90.3% of 506 Tango pairs and 87.4% of 111 SCION pairs provide at least one unique alternative. Both substrates offer a median of two unique alternatives per AS pair, with some providing as many as 11 alternatives in Tango and 10 in SCION (Fig. 4.2a).[2] Moreover, these alternatives are often structurally distinct: among pairs with available alternatives, the best option avoids at least half of the reference path's intermediate ASes for 99.3% of Tango pairs and 59.5% of SCION pairs, while adding zero median AS-hop overhead (Fig. 4.2b–4.2c).

However, raw path diversity only represents an opportunity, not a standalone solution. Candidate paths exhibit differing AS-level overlap, and even rich path sets often contain a common "bottleneck" AS that remains a powerful observer. For instance, only 56.5% of Tango pairs and 20.7% of SCION pairs possess path sets where at least half of the available paths avoid the most common AS (Fig. 4.2d). Consequently, naïve traffic splitting can fail to improve privacy: even when traffic is divided across multiple paths, a worst-case AS may still observe enough of the trace to fingerprint the user.

[1] We define the BGP default path as the reference for Tango, and the most frequently active AS-level path as the reference for SCION.

[2] For SCION, we collapse available interface-level paths into coarser AS-level path sets; as a result, our AS-level diversity estimates are substantially more conservative than SCION's raw interface-level diversity.

The central challenge is therefore one of *allocation*. For any source–destination pair, PraxiGuard must determine the optimal traffic fraction to assign to each path. This decision is fundamentally constrained by the underlying topology: identical allocations can induce vastly different AS exposures depending on path overlap, while different path sets impose varying latency costs. Because exhaustively testing every potential configuration is impractical, PraxiGuard instead models the privacy and performance consequences of candidate allocations to search for an optimal, low-cost split.

To achieve this, PraxiGuard integrates AS-level topology data and per-site empirical mappings into an Integer Linear Programming (ILP) optimizer. Topology determines which AS-level observers see what fraction of the protected traffic under a candidate allocation. Per-site mappings then evaluate that allocation along two dimensions: a privacy mapping translates partial exposure into privacy risk (*e.g.,* predicted WFP classification accuracy), while a performance mapping translates a chosen path-cost summary (*e.g.,* added delay or traffic-weighted added delay) into an application-level cost (*e.g.,* page load time overhead). We empirically derive these relationships by varying partial traffic exposure, added delay (relative to the reference path), and the fraction of traffic assigned to delayed paths, in order to measure site-specific variances in fingerprinting robustness and performance sensitivity. Given these inputs, PraxiGuard first identifies allocations that minimize the strongest observer's predicted accuracy, then chooses the lowest-cost allocation

that satisfies a given performance budget. Finally, the selected allocation is enforced at a flow granularity at the network edge.

We evaluate PraxiGuard across Tango and SCION substrates using real path data and empirical measurements from representative sites (Table 4.2). Compared to an undefended default setting, PraxiGuard reduces worst-case AS-level attacker accuracy by a median of 52 percentage points—with gains reaching 99 pp—while maintaining page load time overheads $\leq 5\%$. On the real-world Tango substrate, PraxiGuard improves privacy for 87% of evaluated source–destination pairs. PraxiGuard also outperforms a naïve uniform splitting baseline in 56% of Tango pairs, confirming that topology-aware allocation is a worthwhile approach for harnessing traffic splitting into measurable privacy gains.

We define PraxiGuard's problem setting, threat model, and design requirements in §4.2, then present the system overview and key design insights in §4.3. §4.4 describes the empirical privacy-performance mappings and ILP formulation, and §4.5 evaluates PraxiGuard across Tango and SCION path substrates. §4.6 includes additional discussion, followed by related work in §4.7 and the conclusion in §4.8.

4.2 PraxiGuard Problem Setting

In this section, we define the setting in which PraxiGuard operates. We first specify our threat model for AS-level WFP mitigation, then describe our guiding design requirements.

4.2.1 Threat Model

PraxiGuard targets network-layer WFP mitigation for traffic exchanged between trusted edge networks which share the same privacy goals. The aim of our work is not to provide the full anonymity guarantees of systems such as Tor, which hide users even from their local network provider. Instead, PraxiGuard targets a complementary setting: users trust their edge network to act on their behalf, but do not trust intermediate Autonomous Systems (ASes) that carry their traffic across the Internet. This setting naturally arises in campus networks, enterprises, ISPs, cloud-edge deployments, and privacy-conscious organizations where a network operator is incentive-aligned with its users.

Clients. Clients are users whose website visits should be protected from AS-level WFP, without imposing high performance costs. Clients send ordinary application traffic through a trusted edge network, such as a campus gateway or enterprise border. Traffic may traverse shared gateways, NATs, or proxies, so external ASes observe traffic associated with the edge network rather than a specific internal host. This provides a limited form of edge-mediated privacy, but it does not eliminate traffic-pattern leakage. Clients do not trust any intermediate ASes, but do trust their local edge network (*i.e.,* their operator), which has incentives to protect their privacy by reducing what intermediate ASes can infer from observable traffic patterns (*e.g.,* a corporation seeking to hide employee access to sensitive web services).

Operators. Operators are edge networks that deploy PraxiGuard for users within their administrative domain; we assume these operators are trusted by, and incentive-aligned with, their users. These operators may include campus networks, enterprises, ISPs, cloud providers, or hosted services; individual users with administrative control over an edge or border router may also act as operators[3]. Operators may be cooperative: privacy-motivated edges share a common goal of reducing exposure to AS-level observers. However, PraxiGuard does not require joint coordination across edges; each operator can *independently* optimize its local traffic allocations to reduce exposure for its respective traffic direction. When available, cooperating edges provide practical advantages, such as exposing path diversity and supporting trustworthy measurements. With Tango, for instance, cooperating edges can help identify candidate interdomain paths and measure the one-way delays added by those paths[28]. We assume operators can measure path latencies and note that high-precision clock synchronization is unnecessary, as PraxiGuard compares allocations using only *relative* path-cost estimates rather than absolute performance metrics.

Adversaries. The adversary is a passive AS-level observer located on one or more paths between the cooperating edges (*e.g.,* see Fig. 4.1a). Each path may traverse one or more intermediate ASes (Fig. 4.1b), all of which are untrusted. That is, any AS may be adversarial, though it is unknown to the defenders which ASes, if any, act adversarially. Accordingly,

[3] For example, in SCION, path selection can be exercised either by native SCION endpoints or by edge-deployed SCION-IP gateways (SIGs).

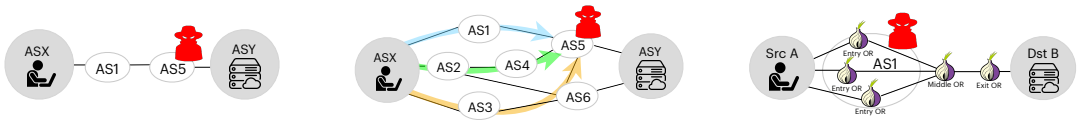


Figure 4.1: AS-level threat model. (a) A single-path setting allows one AS to observe all traffic. (b) Even when traffic is split across multiple network-level paths, all paths may still traverse the same AS (*e.g.*, AS5), degrading traffic exposure to the single-path setting[4]. (c) Traffic splitting across multiple Tor entry guards[53] can similarly concentrate traffic through one AS (*e.g.*, AS1).

PraxiGuard optimizes against the worst-case AS observer: the AS that, under a given traffic allocation, obtains the most useful view of the protected trace.

We model AS-level observability *directionally*, as interdomain routing determines traffic paths independently for each direction. A passive AS observes only the traffic sent over paths that include it, and because forward and reverse AS paths are often asymmetric, they need not involve the same observers. This captures a realistic on-path AS threat, where privacy risk arises from the partial trace exposed to any network carrying the protected traffic. Accordingly, we do not model stronger adversaries that correlate observations across both directions, or across multiple colluding ASes. Like prior work[78,53], we assume a passive adversary capable of monitoring traffic but not decrypting, modifying, or injecting it. We also assume standard encrypted application traffic, so payloads are unavailable to the adversary.

Adversaries can distinguish the start and end of a web visit, separate it from background traffic, and have sufficient resources to train machine learning classifiers on collected traces.

[4]Extracted from a real-world Internet topology (see §4.5.1 for methodology).

Importantly, we assume the adversary is aware of the defense strategy (*i.e.,* the traffic-splitting scheme), and can train on protected traces. Consistent with established evaluation methodology[53,140,78], we adopt a closed-world threat model, where adversaries know the set of possible sites the user may access and aim to identify the specific one. While abstracting away real-world browsing complexity, this setting represents a favorable scenario for the adversary—maximizing their advantage—and allows us to assess our defense under worst-case conditions. We leave extending the evaluation to open-world scenarios, where adversaries contend with greater uncertainty, to future work.

Non-goals and out-of-scope attacks. PraxiGuard does not aim to hide users from their trusted edge network, provide unlinkability against the edge operator, or replace full anonymity systems. It also does not protect against malicious or compromised trusted edges. We do not target active network attacks such as BGP hijacks, interception, packet injection, or routing manipulation. Following our focus on passive WFP, we assume the adversary exploits traffic exposure on the paths offered by the routing substrate but does not actively alter them. This assumption is strongest for SCION, which provides authenticated AS-level path information under cryptographic protection. For BGP-based substrates like Tango, this assumption is naturally weaker; however, by exposing path diversity through the most specific globally routable prefixes (*i.e.,* IPv4 /24s or IPv6 /48s), Tango effectively limits the scope of common subprefix hijacks (§4.6).

4.2.2 Design Requirements

To protect ordinary browsing traffic effectively, WFP mitigation must satisfy three goals: reduce exposure to network-level adversaries, preserve application performance, and avoid requiring users to manage specialized privacy mechanisms. Thus, we identify the following core design requirements.

AS-Aware Privacy. The defense must reduce what on-path AS-level adversaries can infer from traffic patterns. Because the adversary's location is unknown, privacy cannot be assessed from a single assumed vantage point, nor from the number of connections, interfaces, or paths used. Instead, the relevant risk is worst-case AS exposure: how much of the protected traffic remains available to the most exposed AS. A successful WFP mitigation strategy should therefore reduce the useful signal available to that worst-case observer, rather than naïvely spreading traffic from the endpoint.

Performance Preservation. The defense must preserve application-layer performance while reducing fingerprinting risk. A WFP defense intended for ordinary web traffic cannot rely on large amounts of padding, cover traffic, artificial delay, or overlay detours that make client-side page load times noticeably worse. Instead, it must treat performance as a first-class constraint, bounding user-visible overheads such as client-side PLT.

Table 4.1: Comparison of traffic-splitting WFP defenses.

Defense	AS-Level Privacy	Low Overhead	User Accessibility
TrafficSliver [53]	✗	✗	✗
HyWF [78]	✗	✗	✗
COMPs [167]	✗	✓	✗
PraxiGuard	✓	✓	✓

User-Transparent Operation. The defense should not require individual users to understand AS-level routing, select paths, install specialized anonymity tools, or rely on application-specific mechanisms. Instead, the defense should be compatible with ordinary application traffic and should apply consistently once enabled, so that privacy protection does not depend on each user's technical expertise and willingness to run specialized software or manage per-connection choices.

4.2.3 Limitations of Prior Work

While existing defenses introduce promising techniques, none fully satisfy the design requirements outlined (Table 4.1).

Insufficient Protection against AS-Level Adversaries. Prior traffic-splitting defenses reduce the traffic exposed by any single split (*e.g.,* over an overlay circuit, transport connection, or network interface), but generally do not aim to protect against the AS-level adversary defined in §4.2.1 [53,78,167]. In essence, such defenses abstract away the underlying Internet routing topology, thus overlooking real-world adversarial vantage points. Path

splits that appear independent to the endpoint may still traverse a common transit AS. Thus, defenses that split traffic without modeling AS-level path overlap can overestimate the privacy gained from splitting.

As an illustration, the user located in Src A in Fig. 4.1c that is leveraging TrafficSliver[53] to improve her privacy assumes that her circuits are independent. In practice, her traffic is routed through the same AS, which can reassemble and fingerprint her traffic despite splitting. While prior work outside of the WFP setting has explored AS-level threat models (see §4.7), it has largely focused on relay selection in Tor—particularly to prevent a single AS from observing both entry and exit relays, which enables classic traffic-correlation attacks. Such models properly assume that each AS along a path observes all traffic, but restrict the adversary's capabilities to a single end-to-end path. In contrast, we consider AS-level adversaries which are naturally (*i.e.,* because of the way Internet routing works) capable of monitoring and correlating traffic across multiple ostensibly independent paths; we evaluate this risk separately for each direction of communication, since forward and reverse AS paths may involve different observers.

High Performance Cost. Existing defenses impose significant overhead due to their dependence on Tor's overlay network. Approaches such as TrafficSliver[53] and HyWF[78] operate within the Tor ecosystem, leveraging traffic splitting across multiple circuits or network interfaces to obscure traffic patterns. However, Tor's architecture is inherently

constrained by heavyweight costs from relay-based routing, multi-hop encryption, and software processing. These costs may be acceptable for small set of highly motivated users who require strong anonymity, but they are poorly matched to real-time web usage where page-load performance is a primary usability constraint[7,14,143].

Limited Accessibility for Ordinary Users. Existing splitting defenses also place too much burden on the user or require a narrow deployment setting. Some approaches assume end hosts can exploit multiple network interfaces, such as WiFi and cellular, or rely on specialized multihoming support such as MPTCP-capable clients, bridges, or middleboxes[78]. Others depend on particular transport or application mechanisms, such as connection migration [167], or on users choosing and configuring an anonymity overlay[53]. These assumptions limit the population of traffic that can be protected: they depend on user expertise, device capabilities, protocol support, or application behavior.

4.3 PraxiGuard Overview

PraxiGuard protects against AS-level WFP by strategically distributing traffic across diverse network paths. This approach is rooted in the insight that WFP protection depends on whether a traffic allocation effectively reduces exposure to on-path observers. This section highlights the key technical insights that allow PraxiGuard to minimize privacy risk while maintaining low performance overhead.

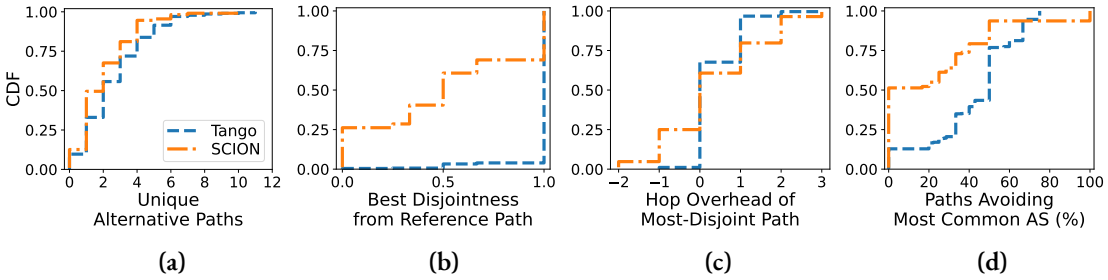
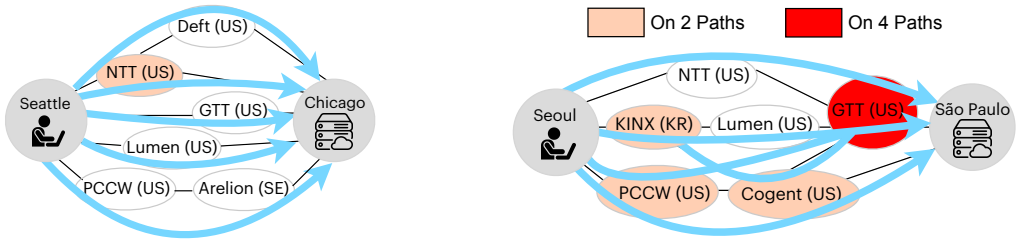


Figure 4.2: Meaningful path diversity is available across many AS pairs using existing Tango and SCION path options. (a) Many AS pairs have a non-trivial number of alternative paths. (b) The most-disjoint alternative often has little AS-level overlap with the reference path. (c) This alternative usually adds little hop overhead. (d) However, available paths can still concentrate on a common AS; creating practical opportunity to reduce worst-case AS exposure through strategic traffic allocation.

PraxiGuard leverages an untapped path-diversity opportunity at the network edge. PraxiGuard is predicated on the fact that edge networks serving everyday users—such as campus, enterprise, or ISP gateways—often have access to substantially more path diversity than conventional routing suggests. We quantify this opportunity by analyzing alternative AS-level paths from two path-control substrates: 506 source–destination AS pairs via Tango, and 111 AS-level SCION pairs. While we reserve the detailed path-collection methodology for §4.5.1, we evaluate this opportunity here across three critical dimensions: path availability, disjointness from the reference path, and AS-hop overhead (Fig. 4.2a-4.2c).

The resulting path sets provide a non-trivial degree of choice at the edge network. Specifically, 67% of Tango pairs and 50.5% of SCION pairs expose at least two unique alternatives, with a substantial fraction—28.1% and 18.9%, respectively—exposing four or more (Fig. 4.2a). Moreover, these alternatives often offer significant structural separa-



(a) High-diversity topology.

(b) Low-diversity topology.

Figure 4.3: Useful path diversity is topology-dependent. Some source-destination pairs offer disjoint AS-level paths, while others repeatedly traverse the same shared ASes.

tion. Among pairs with usable alternatives, 96.1% of Tango pairs and 31% of SCION pairs possess at least one fully reference-disjoint alternative, while 99.3% in Tango and 59.5% in SCION have access to an alternative that avoids at least half of the reference path's internal ASes (Fig. 4.2b). Crucially, this diversity does not automatically increase path length: the most disjoint alternative adds no AS-hop overhead for the majority of pairs (67.6% of Tango and 60.7% of SCION) and introduces at most one additional hop for 96.7% and 79.8% of Tango and SCION pairs, respectively (Fig. 4.2c). Thus, these substrates do not merely increase path counts; they often expose structural routing opportunities that a coordinated edge can potentially leverage to reduce AS-level exposure.

However, this topological opportunity is fundamentally limited. A high count of available alternatives does not guarantee that an edge can safely distribute traffic without explicit reasoning about AS overlap. Many path sets are bottlenecked by a "common" AS that appears across multiple available routes, remaining a dominant worst-case observer despite apparent diversity. Our analysis shows that only 56.5% of Tango pairs and 20.7%

of SCION pairs possess path sets where at least half of the available routes avoid the most common intermediate AS (Fig. 4.2d). Fig. 4.3 illustrates this distinction with representative real-world topologies from Tango. In high-diversity topologies, available paths provide genuine privacy leverage by bypassing different intermediate ASes (Fig. 4.3a). In contrast, low-diversity topologies—where paths converge at a single transit provider—can cause simple path counts to drastically overstate the available opportunity (Fig. 4.3b). Consequently, while available path diversity provides essential leverage to reduce exposure, it is insufficient on its own. To be robust, a defense must move beyond merely splitting traffic over a larger number of paths; it must instead reason about the specific ASes on each path and the exact traffic fraction each observer AS would see under a candidate allocation.

PraxiGuard transforms path exposure into a strategic allocation problem. A central insight of PraxiGuard is the formal separation of path discovery from path exploitation through a traffic-allocation abstraction. Rather than assigning paths a binary "safe" or "unsafe" status, PraxiGuard represents each candidate path set as an AS-level exposure model. A traffic allocation then induces an exposure profile across the network, where an AS's exposure is the total fraction of traffic carried on all traversing paths. This abstraction effectively decouples the underlying path-control substrate (such as Tango or SCION) from the defense. While the substrate enumerates the candidate path set, PraxiGuard determines the optimal weight distribution across those paths to mini-

mize information leakage. Inspired by the approach of oblivious routing—which optimizes for worst-case scenarios without prior knowledge of specific demands—PraxiGuard selects an allocation that reduces the impact of the most powerful observer in any given topology without needing to identify which ASes are adversarial. This modular design allows PraxiGuard to be tailored to different topologies, automatically extracting the maximum available privacy from the specific topological constraints of each path set.

Optimal allocation is fundamentally site-specific. Even within a fixed topology, the "best" traffic split is not universal; rather, it is highly dependent on the traffic patterns and resource dependencies of the specific website being accessed. Privacy leakage is intrinsically tied to a page's unique fingerprint: for some sites, an adversary observing only 40% of the traffic may still maintain high classification accuracy, while for others, the same exposure yields significantly less information (Fig. 4.5). Performance sensitivity is similarly site-dependent, but with even greater variance. The page load time (PLT) of a website is governed by a site's specific connection structure, object dependencies, and loading behavior. Consequently, the same delays experienced across different paths in a particular path set might be negligible for one page but induce prohibitive overhead for another (Fig. 4.7).

These observations lead to a critical design insight: because websites respond differently to traffic splitting, PraxiGuard must move beyond a "one-size-fits-all" network policy to a site-aware optimization strategy. By incorporating the specific privacy and performance

characteristics of individual sites, PraxiGuard can tailor its allocations to the unique requirements of ordinary application traffic. To facilitate this, PraxiGuard employs two key abstractions for comparing candidate splits. *Privacy mappings* relate AS-level traffic exposure to predicted attacker accuracy, capturing the varying resilience of different sites to fingerprinting under partial observation. *Performance mappings* relate a chosen path-cost summary (*e.g.,* added path delay, or the traffic-weighted added delay induced by a split) to a chosen application-level cost metric (*e.g.,* PLT overhead), capturing how user-experienced performance responds to routing decisions. By operating at this site-specific level, PraxiGuard allows an operator to navigate the privacy–performance trade-off using the metrics that matter most: actual privacy risk and user-visible performance cost.

PraxiGuard's end-to-end design. Fig. 4.4 summarizes how PraxiGuard synthesizes these insights into an integrated optimization framework that transforms topological and empirical data into an actionable defense. The optimization itself takes three primary inputs: a candidate set of AS-level paths annotated with relative path-cost estimates, site-specific mappings for both privacy and performance, and high-level operator objectives. PraxiGuard processes these inputs through an ILP-based optimizer and computes a traffic allocation—defined as the percentage of a site's traffic to be routed through each available path. In Fig. 4.4, we show an example objective: minimizing the predicted classification accuracy of the strongest AS observer while maintaining less than 5% performance

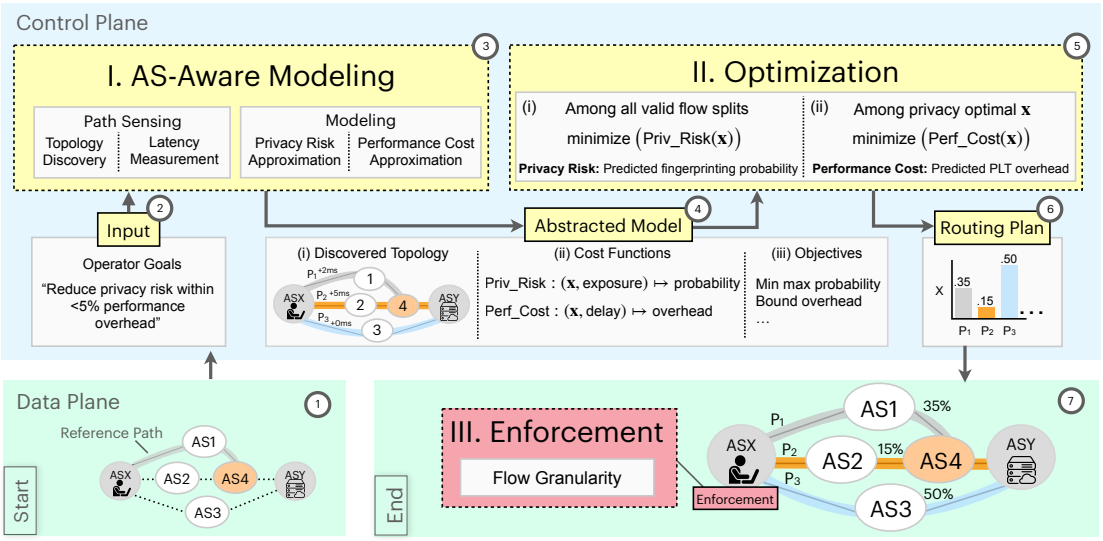


Figure 4.4: PraxiGuard overview. A trusted edge operator (*e.g.*, ASX) discovers available AS-level paths, estimates their relative costs, and combines this topology information with site-specific mappings to compute an optimal allocation plan. The resulting split is enforced at flow granularity, reducing on-path AS-level exposure while respecting operator-defined performance goals.

overhead. This ensures the defense is topology-aware: it selectively exploits path diversity while accounting for real disjointness, effectively increasing privacy against ASes that appear on multiple seemingly distinct routes (§4.5.2). Simultaneously, it remains content-aware, tailoring the split to the unique fingerprint and latency sensitivity of the target website (§4.4.1). Finally, the chosen allocation is enforced at the network edge using flow-level steering. Specifically, the optimizer's weights are installed by the PraxiGuard control-plane as weighted hash-based forwarding rules in the edge switch, so each new flow is assigned to one path while all packets within that flow remain path-consistent. By assigning traffic to paths at the granularity of flows rather than packets, PraxiGuard gains privacy

while avoiding transport-layer performance costs incurred by packet-level splitting (§4.5.3).

4.4 Privacy-Performance Optimization

Building on the architectural insights in §4.3, this section formalizes the exploitation of available network paths for privacy protection as a constrained traffic-allocation problem. In this framework, PraxiGuard must determine the optimal distribution of traffic across a candidate set of AS-level paths to maximize privacy while satisfying strict performance guarantees. An allocation $\mathbf{x}$ (the vector of traffic weights assigned to each path) dictates the two primary metrics of interest: the privacy risk induced by observation at common ASes, and the performance cost resulting from path-level latencies. Because these metrics are sensitive to both network topology and the specific behavior of the target application, PraxiGuard cannot rely on static heuristics. Furthermore, since exhaustively measuring the impact of every possible allocation in real time is computationally and operationally infeasible, PraxiGuard utilizes empirical mappings to approximate these outcomes. While the detailed measurement methodology for constructing these mappings is deferred to §4.5.1, the following sections define the formal abstractions and the optimization objective used to compute the final traffic policy.

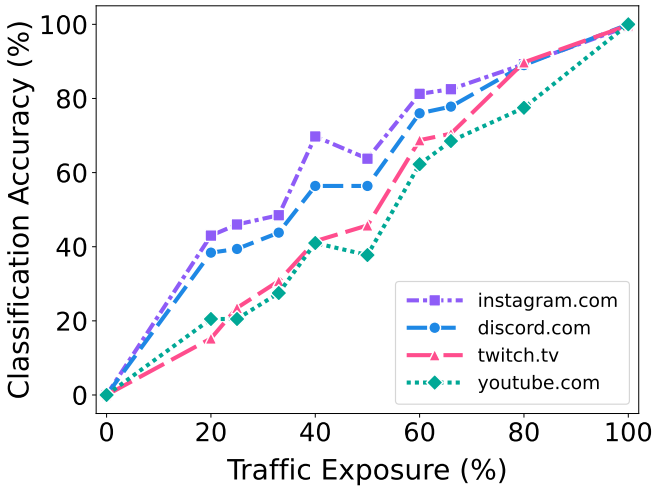


Figure 4.5: Privacy risk mappings for representative sites. Each curve maps the fraction of traffic exposed to an AS-level adversary to the resulting website-fingerprinting classification accuracy. PraxiGuard uses these empirical mappings to estimate privacy risk under different traffic-splitting decisions.

4.4.1 Empirical Privacy and Performance Mappings

Privacy risk mappings

For a given website s, PraxiGuard models the privacy risk through a mapping function $f_s(v)$ that relates the fractional AS-level traffic exposure v to predicted website-fingerprinting classification accuracy. Traffic exposure is a useful intermediate representation because it is both topology-derived and classifier-relevant: a candidate allocation directly determines what fraction of the trace each AS observes, and the empirical mapping can then translate that fraction into expected WFP accuracy for each site. This abstraction allows the optimizer to quantify the utility of partial observability; a traffic split is only considered effective if it can significantly degrade the adversary's classification accuracy.

To construct these mappings, we employ LaserBeak[106], a state-of-the-art transformer-based WFP classifier. By utilizing a transformer architecture, the classifier can capture complex, long-range dependencies in packet sequences, making it a highly robust baseline for evaluating privacy. We empirically generate the mappings by simulating an adversary who observes varying fractions of a site's traffic—ranging from 0% to 100%—and recording the resulting classification accuracy.

As illustrated in Fig. 4.5, while accuracy generally scales with increased exposure, the sensitivity of this relationship is highly site-dependent. For instance, at 40% traffic exposure, an adversary can achieve nearly 70% accuracy for instagram.com, whereas the same exposure for youtube.com yields only approximately 40% accuracy. These disparities indicate that certain site fingerprints are more resilient to partial observation than others and that a "universal" privacy curve might potentially under-protect more sensitive sites.

Performance cost mappings

Like privacy risk, performance cost varies across sites and must be characterized empirically. However, while privacy risk possesses a natural intermediate representation in AS-level exposure, the performance equivalent is less immediate. For example, PLT is governed not just by path latencies, but also by browser-side factors—such as request scheduling and inter-resource dependencies—where delaying one resource can block subsequent requests and stall page rendering. Instead of modeling all these dynamics directly, PraxiGuard

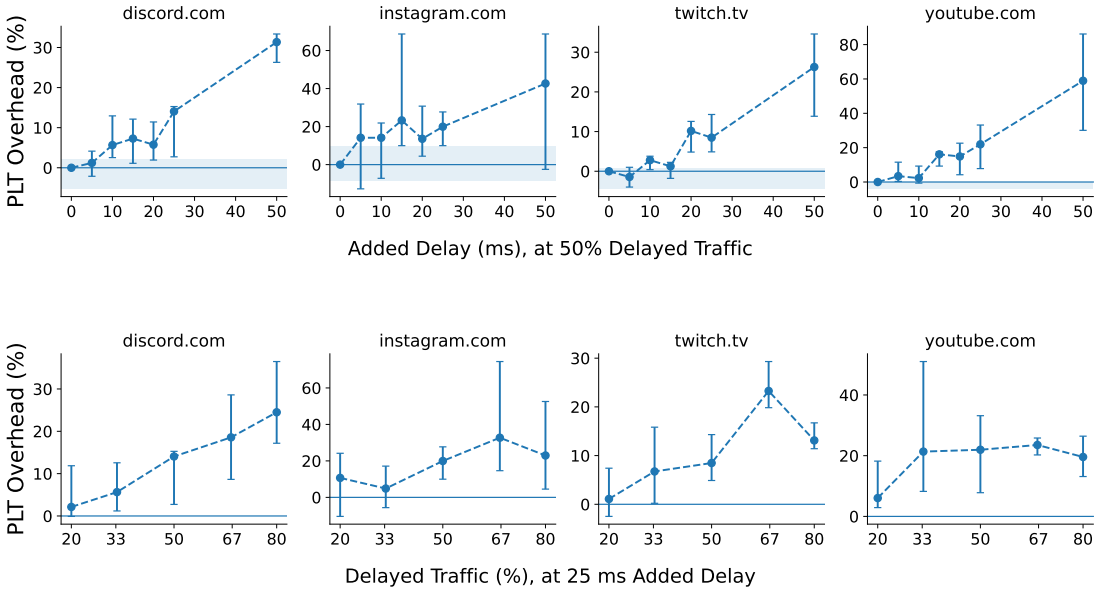


Figure 4.6: Path delay and delayed-flow fraction both affect page-load performance, but the effect is site-specific. Top: PLT overhead as added path delay increases while delaying 50% of downstream flows. Bottom: PLT overhead as the nominal delayed-flow fraction increases with 25 ms added delay. Points show median PLT overhead across post-warm-up visits; error bars show the interquartile range. The shaded band in the top row shows zero-delay baseline variation across split configurations. These measurements motivate empirical per-site performance mappings rather than a single aggregate latency model.

uses empirical measurements to construct a compact mapping from allocation-level path costs to application-level performance impact, such as PLT.

To construct one such representation, we isolate the sensitivity of PLT to two allocation-level factors: the estimated added cost d_p of a path relative to the reference path, and the fraction of traffic assigned to that path x_p. As illustrated in Fig. 4.6, both factors significantly influence PLT overhead, though the magnitude of the effect varies significantly across sites. For example, youtube.com exhibits a sharp sensitivity to added delay (top row), with overhead reaching nearly 60% at 50ms of delay, whereas twitch.tv remains more re-

silient, staying below 30% overhead for the same delay. Similarly, the fraction of traffic delayed (bottom row) shows that for sites like discord.com, increasing the delayed-flow fraction from 20% to 80% results in a steady, near-linear overhead increase, while instagram.com displays high sensitivity even at low fractions.

PraxiGuard therefore adopts *traffic-weighted added delay* ($\bar{d}$) as an intermediate path-cost summary for estimating performance cost. As established in §4.2.1, these costs can be supplied by the path-control substrate or by cooperating-edge measurements; PraxiGuard only requires relative cost estimates for comparing allocations. For an allocation $\mathbf{x}$ over candidate paths $\mathcal{P}$, where x_p is the fraction of traffic assigned to path p and d_p is the estimated cost of path p relative to the reference path, we define traffic-weighted delay as:

$$\bar{d} = \sum_{p \in \mathcal{P}} x_p d_p. \tag{4.1}$$

This metric effectively collapses the two critical factors identified in our measurements: delay magnitude and traffic fraction.

Fig. 4.7 shows the resulting performance mappings used by the solver to evaluate potential allocations. Each empirical point represents the median PLT overhead for measured configurations with the same traffic-weighted added delay, computed across 20 repeated visits (following an initial warm-up visit). Although the overall trend generally increases with $\bar{d}$, visit-level variability can produce local non-monotonicity, where configurations

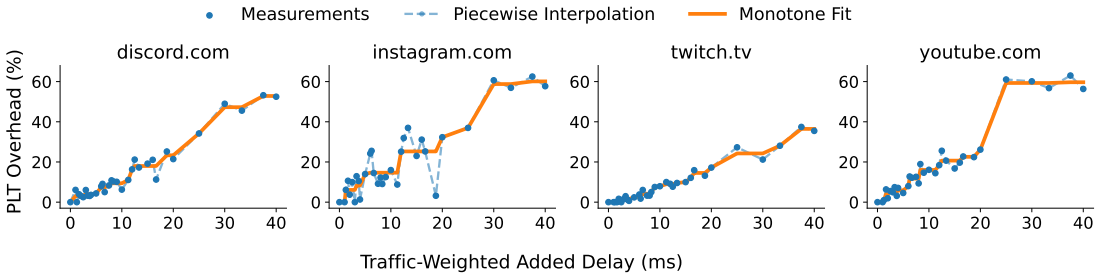


Figure 4.7: Per-site performance mappings. We summarize each split configuration by its traffic-weighted added delay and map it to median PLT penalty. Dots show empirical measurements of median PLT overhead after grouping by traffic-weighted added delay; negative changes are clipped to zero for the solver-facing penalty. Dashed lines show piecewise interpolation between measured points, and solid lines show the monotone fit used by the solver. The fitted mapping provides a conservative performance guardrail when selecting privacy-improving traffic splits.

with higher weighted delay appear to yield slightly lower median overheads. Thus, to provide a stable performance constraint for the optimizer, we conservatively fit an isotonic, monotone piecewise-linear mapping to the empirical measurements.

4.4.2 ILP Formulation

PraxiGuard operationalizes the relationship between network topology and application-layer outcomes by modeling traffic allocation as a constrained optimization problem. For a given site s and source–destination pair, the goal is to determine an allocation $\mathbf{x}$—the vector of traffic weights assigned to each candidate path $p \in \mathcal{P}$—that optimizes a chosen privacy objective while adhering to a performance envelope.

The Feasible Allocation Space

The feasible space is defined by three requirements: the allocation must satisfy traffic conservation (assigning all traffic across available paths), the induced AS-level exposure must be topologically consistent, and the predicted performance cost must be remain the operator's budget. Let $\mathcal{A}$ denote the set of internal ASes that appear on at least one candidate path, and let $\mathcal{P}_a \subseteq \mathcal{P}$ denote the subset of paths that traverse AS a. Each path p has an added delay d_p relative to the reference path. Given a PLT-overhead budget B, an allocation is feasible if it satisfies:

$$\sum_{p \in \mathcal{P}} x_p = 1, \tag{4.2a}$$

$$x_p \geq 0 \qquad \forall p \in \mathcal{P}, \tag{4.2b}$$

$$v_a = \sum_{p \in \mathcal{P}_a} x_p \qquad \forall a \in \mathcal{A}, \tag{4.2c}$$

$$r_a = f_s(v_a) \qquad \forall a \in \mathcal{A}, \tag{4.2d}$$

$$g_s\left(\sum_{p \in \mathcal{P}} x_p d_p\right) \leq B. \tag{4.2e}$$

Constraints (4.2a) and (4.2b) ensure that $\mathbf{x}$ is a valid traffic distribution: all traffic is assigned to candidate paths, and no path receives a negative traffic fraction. Constraint (4.2c)

computes the exposure v_a at each AS a as the aggregate traffic fraction of all paths intersecting that AS. Constraint (4.2d) maps this exposure through the site-specific privacy mapping $f_s(\cdot)$ to yield the predicted classification accuracy r_a for AS a. Finally, Constraint (4.2e) enforces the performance envelope by mapping traffic-weighted delay through the site-specific function $g_s(\cdot)$. By evaluating the formulation across a range of values for B, PraxiGuard generates a privacy–performance frontier that allows an operator to select an optimal split within their overhead tolerance.

Optimization Objectives

Given the feasible space, PraxiGuard supports multiple privacy objectives over the vector of per-AS risks $\mathbf{r} = \{r_a\}_{a \in \mathcal{A}}$.

Minimizing the strongest observer. To defend against worst-case adversaries, PraxiGuard adopts a minmax strategy:

$$\min_{\mathbf{x}} \quad \max_{a \in \mathcal{A}} r_a \quad \text{s.t. constraints (4.2).} \tag{4.3}$$

This objective minimizes the maximum classification accuracy achievable by any single intermediate AS, neutralizing the most powerful observer in the topology.

Reducing the attack surface. Alternatively, PraxiGuard can minimize the total number of high-risk AS observers whose predicted accuracy exceeds a threshold τ:

$$\min_{\mathbf{x}} \quad \sum_{a \in \mathcal{A}} \mathbb{I}[r_a > \tau] \quad \text{s.t. constraints (4.2).} \tag{4.4}$$

This policy thins the attack surface by reducing the count of adversaries capable of high-confidence fingerprinting. In the MILP implementation, we linearize the indicator using a binary variable $z_a \in \{0, 1\}$ for each AS and a sufficiently large constant M (where $M = 100 - \tau$):

$$r_a \leq \tau + M z_a \quad \forall a \in \mathcal{A}, \tag{4.5}$$

Minimizing $\sum z_a$ thus minimizes the number of observers for whom $r_a > \tau$.

Implementation and Output

To ensure compatability with standard MILP solvers, PraxiGuard represents the empirical mappings $f_s(\cdot)$ and $g_s(\cdot)$ as piecewise-linear functions. We linearize the minmax objective by introducing an auxiliary variable $\Phi_{\max}$ and imposing $r_a \leq \Phi_{\max}$ for all $a \in \mathcal{A}$. The optimizer's final output is a discrete path-level traffic allocation, which can then be realized by an edge-steering mechanism. §4.5.3 details how we utilize flow-level steering to preserve these privacy gains while maintaining performance.

4.5 Evaluation

We evaluate whether PraxiGuard can leverage existing path diversity to provide practical protection against AS-level website-fingerprinting attacks. The preceding sections established the two foundational inputs required for our optimization framework: path availability and privacy-performance mappings. Specifically, §4.3 demonstrated that meaningful AS-level path diversity is made available by systems like Tango and SCION, while §4.4 detailed how PraxiGuard transforms these path choices and their associated costs into optimized traffic allocations. Building on this, we now describe our data collection methodology and experimental setup. We then evaluate whether PraxiGuard effectively reduces worst-case AS inference risk, eliminates high-risk AS observers, and remains effective under flow-level enforcement.

Our evaluation is thus structured in three parts: *Experimental Inputs* (§4.5.1): We describe the construction of the experimental inputs used throughout this study, including the Tango and SCION AS-level path sets and the site-specific privacy and performance mappings consumed by the optimizer. We also describe the construction of aggregate mappings, which average these empirical per-site measurements to serve as a site-agnostic baseline. *Optimization Gains* (§4.5.2): We demonstrate that, within a minimal predicted PLT overhead budget, PraxiGuard significantly reduces worst-case attacker accuracy and eliminates high-risk AS observers compared to both undefended traffic and a uniform

splitting defense. We further compare per-site versus aggregate-site mapping allocations to quantify when per-site optimization provides additional privacy benefit. *Enforcement Granularity* (§4.5.3): We show that optimized allocations can be enforced at flow granularity, preserving the privacy benefits of reduced traffic exposure while avoiding reordering penalties caused by packet-splitting.

4.5.1 Experimental Inputs

Path Diversity Substrates. We evaluate PraxiGuard using two complementary path substrates, which both provide meaningful AS-level diversity (§4.3). For *Tango*, we utilize a live Vultr deployment spanning 23 source and 22 destination nodes (506 pairs). For each pair, we employ Tango's community-suppression technique to uncover alternate BGP-compliant forwarding paths that would otherwise remain hidden under standard BGP operation. Each route is converted into a unique AS-level path, with the default BGP route serving as the baseline reference. For *SCION*, we extract paths from a public SCION measurement database [169]. We parse source-destination SCION address pairs and collapse SCION's interface-level paths into unique AS-level path sets. Since SCION does not have a single "default" route, we use the most frequently observed path for each pair as the reference path. For both substrates, we focus on intermediate AS observers by excluding the source and destination ASes. We omit path sets where no internal AS observers exist, as these are inherently private under our threat model.

Table 4.2: Sites used in our empirical measurements, with median number of flows and baseline TCP handshake RTT.

Domain	Median Flows	Baseline RTT (ms)
amazon.com	114	23.24
discord.com	11	6.31
instagram.com	14	7.31
linkedin.com	38	8.36
netflix.com	68	8.48
pinterest.com	36	7.38
spotify.com	110	8.15
twitch.tv	56	7.38
wordpress.com	37	8.05
youtube.com	22	9.02

Site-Specific Privacy Exposure Mappings. We used a Playwright-based headless browser to collect traffic traces for a diverse subset of the Alexa Top 100 websites. We performed 50 visits per site, capturing packet-level metadata via `tshark`. To ensure sufficient density for realistic multipath splitting, we filtered the dataset for reachability and a minimum volume threshold of 250 packets. This yielded a dataset of 4,150 complete traces across 85 distinct domains. While we verified our findings across the full 85-site dataset to ensure generalizability, we focus our presented analysis on the ten representative sites in Table 4.2 to allow for a more in-depth evaluation of our site-specific optimization.

Mapping Generation: We simulate varying levels of adversarial exposure by splitting these traces according to a Dirichlet distribution ($\alpha = 0.5$). To evaluate the privacy risk associated with these splits, we utilize *Laserbeak*, a state-of-the-art, transformer-based website fingerprinting classifier. While we evaluated several attack classifiers, Laserbeak consis-

tently proved the most effective, achieving 97% accuracy on undefended traces and providing a robust upper bound on adversarial capability. For each site, we split traffic across m paths and measure classification accuracy when the adversary observes n of them, varying $m \in [1, 5]$ and $n \in [1, \min(4, m)]$.

Site-Specific Performance Cost Mappings. We used the same Playwright-based automated browser framework to collect live performance metrics while visiting ten popular websites (listed in Table 4.2) from a dedicated Ubuntu 22.04 test server. Each site was visited 20 times, and standard client-side performance metrics—including Time to First Byte (TTFB), DOMContentLoaded, and WindowLoaded (i.e., PLT)—were recorded for each session. To mitigate cold-start effects and ensure consistent measurement conditions, we performed a warm-up visit for each site prior to data collection, allowing browser caches and DNS resolution to stabilize. To isolate the impact of network delay on application performance, we conducted experiments in a controlled environment using a programmable switch capable of fine-grained, per-packet delay injection. Delay was implemented via a hardware-level "jail-lock" timestamping mechanism: upon arrival, each packet was timestamped, assigned a target release time based on the desired *ms* of delay, and recirculated within the switch until its release time, after which it was forwarded to the end host.

Mapping Generation: We introduced artificial delay across the same (n, m) traffic configurations used in the privacy analysis, to emulate network delay conditions commonly

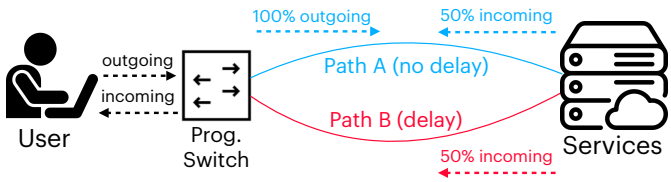


Figure 4.8: Experimental delay setup for controlled end-to-end performance measurements.

observed in the wild, while eliminating external variability that could skew results. Fig. 4.8 illustrates the setup for the $(n, m) = (1, 2)$ case, where 50% of incoming traffic is delayed.

4.5.2 Optimization Gains

We evaluate whether PraxiGuard effectively translates path diversity and mapping inputs into tangible privacy gains across two key dimensions: worst-case adversarial accuracy and the size of the AS attack surface. For each source–destination AS pair, the PraxiGuard optimizer selects a traffic allocation across available Tango or SCION path options to minimize these adversarial risks within a minimal (*i.e.,* 5%) predicted PLT-overhead budget. We compare PraxiGuard against two baselines. The first is *no defense*, where traffic follows a single reference path, allowing any internal AS on that path to observe the full trace. The second is *uniform splitting*, where traffic is divided equally across all available paths, without explicitly accounting for path-specific privacy or performance costs. We define privacy gain as the reduction in worst-case predicted attacker accuracy (in percentage points) relative to these baselines. Additionally, we evaluate the attack surface reduction by measuring the number of "high-risk" ASes (achieving $\geq 50\%$ accuracy) avoided by our optimization.

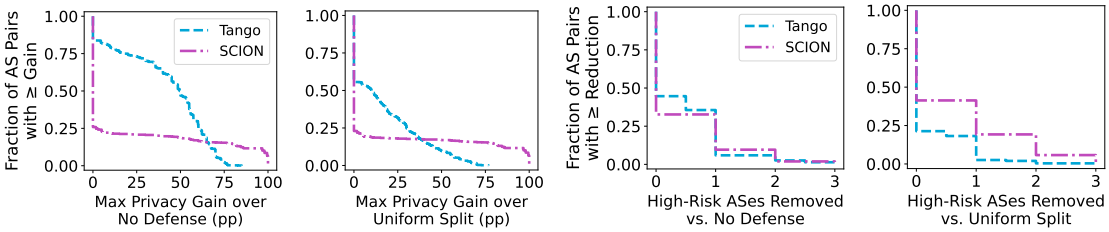


Figure 4.9: Optimization gains: under a 5% PLT-overhead budget, PraxiGuard reduces worst-case attacker accuracy compared to (a) no defense and (b) uniform splitting. Beyond reducing the strongest observer's accuracy, PraxiGuard also reduces the number of high-risk AS observers over (c) no defense and (d) uniform splitting.

How well can PraxiGuard protect against the strongest AS observer? As shown in Fig. 4.9(a)–(b), PraxiGuard consistently reduces worst-case AS attacker accuracy. Compared to no defense, PraxiGuard improves privacy for 87.2% of Tango AS pairs, with a median gain of 50.7 percentage points; for SCION, gains are less broad but can be high-impact, reaching up to 86.1 percentage points. Notably, these gains are immediately practical: PraxiGuard leverages existing path diversity from these networks without requiring new client-side mechanisms or user configuration. Importantly, PraxiGuard also outperforms uniform splitting, particularly with Tango path options. This distinction highlights the specific value of PraxiGuard's optimization: rather than merely spreading traffic, PraxiGuard intelligently identifies allocations that minimize exposure to the strongest potential observers. By shifting traffic away from high-risk ASes, PraxiGuard recovers privacy that uniform splitting leaves unrealized, while strictly respecting the performance budget.

Can PraxiGuard reduce the AS attack surface? Beyond weakening the single strongest observer, we evaluate PraxiGuard's ability to shrink the overall "attack surface"—defined as the number of internal AS observers capable of achieving at least some threshold of predicted accuracy. Fig. 4.9(c) and (d) demonstrate our results for a threshold case of 50%. Compared to no defense, PraxiGuard removes at least one high-risk observer for roughly 45% of Tango pairs and 33% of SCION pairs, with a smaller tail removing two or more ASes. Even relative to uniform splitting, PraxiGuard removes additional high-risk observers for about 22% of Tango pairs and 42% of SCION pairs, showing that optimized allocation lowers the ceiling of adversarial capability while simultaneously reducing the number of ASes that remain effective observers.

Does site-specific modeling improve optimization? Because our empirical measurements reveal significant heterogeneity in how different websites respond to both traffic splitting and network delay, we investigate whether capturing these site-level nuances improves optimization outcomes. We compare the privacy gains achieved by PraxiGuard using site-specific mappings against a baseline where the optimizer utilizes a single aggregate-mean mapping derived from combined domain measurements. As illustrated in Fig. 4.10, privacy gains obtained via per-site mappings match (within 1 pp) or exceed those of the aggregate baseline across nearly all AS pairs for both path substrates. For SCION, the two approaches are mostly comparable. For Tango, the benefits are more pronounced,

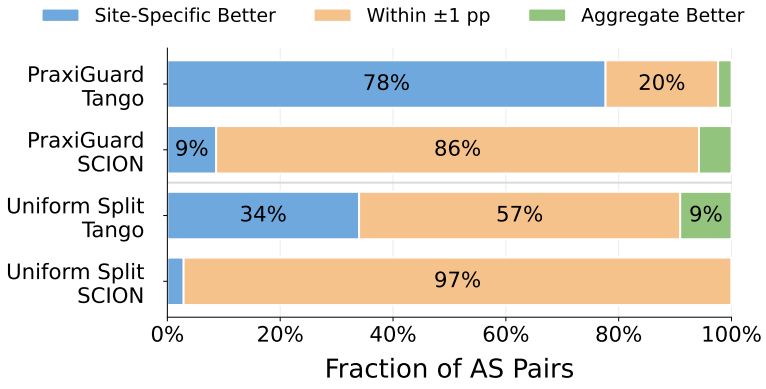


Figure 4.10: We compare privacy gains achieved, showing the fraction of AS pairs where site-specific mappings improve, match (within ± 1 pp), or underperform compared to aggregate mappings. Site-specific modeling usually matches or outperforms aggregate models, and are particularly effective for PraxiGuard on the Tango substrate.

where per-site mappings improve the optimized split for 77.7% of AS pairs and add 7.2 percentage points of privacy gain on average over the aggregate mapping. These results uncover an important optimization opportunity: site-specific empirical calibration directly affects which allocations PraxiGuard selects, enabling it to recover privacy gains that aggregate models would otherwise miss.

4.5.3 Enforcement Granularity

Finally, we ask how a deployment should realize the optimizer's fractional path allocations. Packet-level splitting assigns individual packets to paths and can closely approximate arbitrary allocations, but it causes packet reordering with paths under different delay conditions. Flow-level splitting assigns each transport flow to one path, preserving transport-layer coherence while approximating the target traffic allocation over the aggregate of many

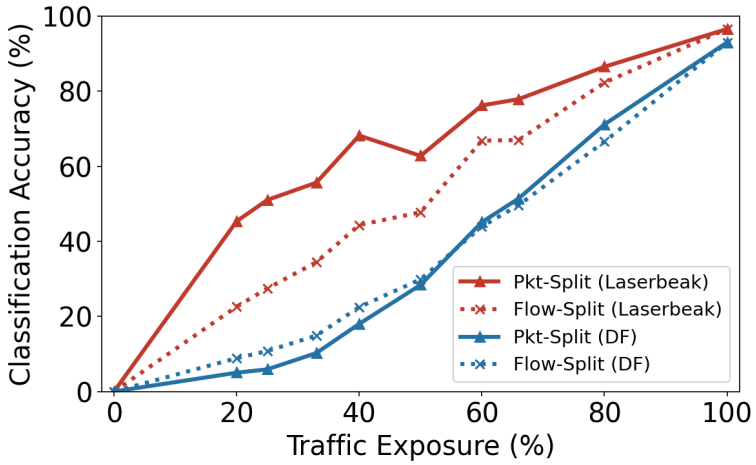


Figure 4.11: Against two different state-of-the-art classifiers, flow-level splitting preserves—and in some cases improves upon—privacy gains compared to packet-level splitting.

flows. We evaluate classification accuracy and PLT induced by both these strategies to determine the more effective enforcement granularity.

Does splitting granularity affect the strength of the privacy gain? We evaluate whether the fine-grained interleaving of packet-level splitting provides a superior privacy signal compared to the coarser flow-level approach. As shown in Fig. 4.11, we compare classifier accuracy across exposure levels for both strategies under the LaserBeak classifier and another state-of-the-art Deep-Fingerprinting (DF) classifier[140], which uses only directional sequences as input. Across tested classifiers, flow-level splitting achieves accuracy reductions comparable to—and in several configurations better than—packet-level enforcement. For LaserBeak, flow-level splitting lowers accuracy by roughly 24 percentage points at 40% exposure (44% vs. 68%) and 9 percentage points at 60% exposure (67% vs.

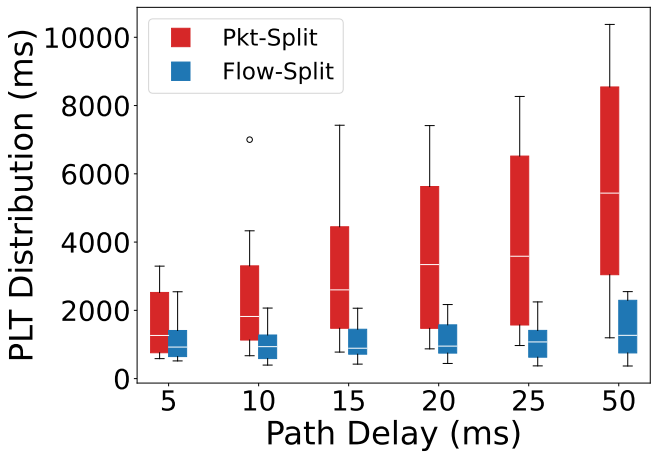


Figure 4.12: Flow-level splitting avoids packet-level PLT penalties. Packet-level splitting creates large and variable PLT inflation as path delay increases, while flow-level splitting keeps page loads substantially closer to baseline.

76%); for DF, the two strategies remain closely matched, with both near 45% accuracy at 60% exposure. These results suggest that, for the classifiers we evaluate, the dominant privacy benefit comes from limiting how much of the trace any observer receives, rather than from the finer-grained packet interleaving provided by packet-level splitting.

How does splitting granularity impact application-layer performance?

While both strategies offer similar privacy protections, their impact on application performance varies significantly when traffic is split across multiple paths. We quantify the PLT overhead incurred by both enforcement granularities using the controlled delay-injection setup described in §4.5.1. Fig. 4.12 reveals a stark contrast: packet-level splitting imposes seconds of additional delay, while flow-level splitting remains much closer to baseline. At 25 ms of added path delay, packet-level splitting raises median PLT overhead to roughly

3.6 s, compared to about 1.0 s for flow-level splitting; at 50 ms, the median packet-level overhead grows to about 5.4 s, while flow-level splitting remains near 1.1 s. Packet-level splitting also exhibits much higher variability, with the upper range exceeding 10 s at 50 ms —an unreasonable cost for ordinary page loads—whereas flow-level splitting remains below roughly 2.5 s. This overhead arises because packet-level splitting breaks the sequence of individual transport flows, forcing receivers to buffer out-of-order packets and stalling the loading of web objects that are sensitive to TCP reordering. In contrast, flow-level splitting keeps each connection on a consistent path, effectively insulating the application from these transport-layer costs.

Together, these results show that flow-level splitting is the optimal enforcement granularity for PraxiGuard. It successfully realizes concrete privacy gains while ensuring the transport layer remains robust to underlying network conditions.

4.6 Discussion

Our analysis assumes that the AS-level paths exposed by the underlying path-control substrate match the paths that packets actually traverse. Active routing attacks violate this assumption: an adversary could try to attract, detour, drop, or selectively manipulate traffic so that the defender's estimated AS exposure no longer matches the realized exposure. This is a different problem from passive WFP because the adversary is no longer merely observ-

ing the allocation chosen by PraxiGuard, but is actively changing the network conditions under which that allocation is evaluated. Addressing this setting would require coupling PraxiGuard with routing-security mechanisms and data-plane validation: the optimizer would need either authenticated path guarantees, as in SCION, or measurements/attestations that detect when BGP-based forwarding deviates from the expected AS path. For Tango-like deployments, prefix granularity can limit simple more-specific subprefix hijacks, but it does not address exact-prefix hijacks, route leaks, malicious forwarding, or control/data-plane inconsistencies. Thus, active routing attacks are best viewed as complementary to our passive AS-level WFP threat model: PraxiGuard reduces information leakage given a set of available paths, while routing security determines whether those paths can be trusted under adversarial manipulation.

4.7 Related Work

WFP attacks exploit metadata from encrypted traffic–such as packet sizes, directions, and inter-arrival times–to infer a user's browsing activity. Early attacks relied on classical machine learning (*e.g.,* k-FP[77], CUMUL[117]), while recent deep learning approaches, notably Deep Fingerprinting (DF)[140], have significantly increased attack accuracy, achieving over 98% accuracy in closed-world settings. More recent work has further strengthened WFP attacks by leveraging limited observations[141], targeting early-stage traffic[56], shifting to Tor

exit-node-based attacks[45], and using transformer-based classifiers as in LaserBeak[106].

Defenses against WFP fall into two main categories: additive obfuscation and traffic splitting. Additive defenses modify traces through padding, delays, or dummy packets. Fixed-rate schemes (BuFLO, Tamaraw[34]) successfully obscure traffic features with these techniques, but incur substantial bandwidth and latency overheads. Probabilistic schemes like WTF-PAD[85] reduce overhead by opportunistically masking traffic gaps, but they are vulnerable to modern deep-learning classifiers. FRONT and GLUE[71] shift focus to early-trace or inter-trace obfuscation to limit cost. QCSD[142] operates at the user-space QUIC layer, shaping traffic patterns in a directional manner. Despite reducing costs, these additive approaches remain brittle against deep-learning attacks.

Traffic-splitting aims to distribute flows across multiple paths, obfuscating fingerprints without injecting overhead. TrafficSliver[53] splits packets across Tor entry guards; Henri et al.[78] leverage ISP multihoming; CoMPS[167] exploits QUIC migration. These approaches reduce WFP accuracy at lower overhead costs than additive defenses, but most of these works rely on Tor, which comes at a significant performance cost itself and cannot support modern browsing speeds. Prior works overlook routing realities and treat all paths as equally viable–applying uniform splitting without regard to AS-level overlap. These abstractions limit their real-world deployability, especially in the presence of Internet centralization, where multiple nominal paths often converge through shared ASes. Finally, works like[167]

are tied to the QUIC protocol, limiting broad usability for different applications.

Prior work on AS-aware path selection in Tor has focused on classic traffic-correlation attacks, aiming to prevent a single AS from observing both the entry and exit of a circuit [149]. These approaches assume adversaries only observe single end-to-end paths and are tightly coupled to Tor's overlay and relay-selection policies. While effective for protecting Tor circuits, they do not address stronger AS-level adversaries capable of correlating traffic across multiple paths, and thus provide limited benefit in our setting. In contrast, we consider a more powerful adversary and design traffic-splitting strategies that leverage Internet-scale AS path diversity beyond Tor. To our knowledge, our work is the first to combine AS-aware traffic splitting with formalized optimization techniques to explicitly target WFP defenses across existing Internet path options.

4.8 Conclusion

PraxiGuard is the first principled framework to provide practical, network-aware website fingerprinting defenses, leveraging AS-level path diversity for policy-driven traffic splitting. Unlike prior approaches, PraxiGuard uses an ILP-based optimizer grounded in per-site empirical models to quantify how traffic allocation affects both worst-case risk and page-load performance. The optimizer then selects privacy-maximizing allocations and, among them, chooses low-cost splits that satisfy a specified performance budget.

Our evaluations with Tango and SCION demonstrate that under low PLT overheads ($\leq$5%), PraxiGuard reduces worst-case attacker accuracy by a median of 52 percentage points (up to 99 pp) and improves privacy for 87% of studied Tango AS pairs. These results reveal a key opportunity: on real-world path substrates with heterogeneous latencies, strong privacy gains need not necessitate severe performance penalties. Thus, we show that optimized, network-aware traffic allocation can systematically harness existing path diversity and offer a path toward practical, performant WFP protection at Internet scale.

5

Conclusion

The modern Internet leaves users exposed to persistent threats to security, privacy, and performance. Although these vulnerabilities arise in different forms, they share a common architectural root: the Internet provides limited practical control over how traffic is admitted to and routed across heterogeneous, independently administered domains.

5.1 Summary of Contributions

This dissertation addresses these limitations by strengthening two critical levers at the network edge: ingress control and route control. The central thesis is that stronger protections become possible through cooperative, edge-centered co-design. Rather than building defenses as standalone solutions, the systems developed here coordinate heterogeneous elements to realize more effective defenses that cannot otherwise be achieved in isolation.

Concretely, this dissertation makes the following contributions. We demonstrate that ingress control can be strengthened through secure coordination between high-speed dataplane mechanisms and exact host-side processing. This layered co-design enables the edge to withstand massive resource-exhaustion attacks without sacrificing user performance. We further show that route control can be strengthened by exposing and exploiting alternative interdomain paths within the constraints of existing protocols, and by dynamically allocating traffic across these paths, to jointly optimize for privacy and performance.

Together, these contributions demonstrate that distributed defense systems, when designed to operate as a unified system, can restore meaningful control in ways that are both principled and deployable, protecting everyday users within the Internet as it exists today.

5.2 Lessons Learned

This dissertation suggests that stronger Internet defenses do not necessarily come from pushing more functionality to end hosts or from assuming control over the entire network path. Instead, meaningful gains can come from carefully leveraging the limited but strategic control available at the edge. Across both ingress control and route control, the network edge provides a practical vantage point for acting on traffic before it reaches downstream users and for influencing how traffic enters or traverses the wide area.

A related lesson is that limited control can often be amplified through careful system design. Within a single edge, layering hardware and software enables defenses that are both fast and precise, despite the constraints of each platform in isolation. Across multiple edges, selective cooperation enables new routing options without requiring global deployment or changes from the Internet core. In both cases, the key is not full control, but how to expand what is achievable from the control that is already available.

More broadly, the systems in this dissertation show that Internet defenses must be designed around real operational constraints, not apart from them. Constraints such as partial adoption, limited trust, platform heterogeneity, and performance sensitivity are not peripheral details; they shape what forms of protection are actually viable. Treating these constraints as first-class concerns leads more deployable systems, that ultimately translate to more effective solutions.

5.3 Concluding Remarks

We envision a next generation of network defenses that are no longer hand-partitioned across heterogeneous components, but instead are *automatically synthesized* from high-level security and performance objectives into coordinated implementations spanning switches, servers, and end hosts. In this vision, defenses would combine centralized reasoning with distributed enforcement, compiling abstract policies into concrete placements, rules, and coordination patterns that respect the capabilities and constraints of each platform. Our position paper Sieve [174] sketches an initial direction for this line of work, but realizing this broader vision will require new abstractions and tighter integration across networking, systems, programming languages, and compiler design.

Taken together, this dissertation establishes a practical agenda for edge-centered co-design and cooperation to better protect users in today's Internet. Although the Internet was not designed for strong control over traffic admission and routing in adversarial settings, this work demonstrates that such control can be incrementally strengthened without major infrastructure upgrades. By coordinating mechanisms at and across the network edge, we show that it is possible to improve users' security, privacy, and performance while remaining grounded in the practical realities of today's networked systems.

Bibliography

[1] AARNet (2023). The rise of DDoS attacks in 2023: what you need to know . https://www.aarnet.edu.au/the-rise-of-ddos-attacks-in-2023-what-you-need-to-know#. Accessed: 2023-05-31.

[2] Akamai (2022). SureRoute. https://developer.akamai.com/article/sureroute.

[3] Akella, A., Maggs, B., Seshan, S., & Shaikh, A. (2008). On the performance benefits of multihoming route control. *IEEE/ACM Transactions on Networking* (*TON*), 16(1), 91–104.

[4] Akella, A., Seshan, S., & Shaikh, A. (2004). Multihoming Performance Benefits: An Experimental Evaluation of Practical Enterprise Strategies. In *USENIX Annual Technical Conference, General Track*.

[5] Alhilal, A., Braud, T., Han, B., & Hui, P. (2022). Nebula: Reliable Low-latency Video Transmission for Mobile Cloud Gaming. In *Proceedings of the ACM Web Conference* (pp. 3407–3417).

[6] Almeida, P. S. (2022). A case for partitioned bloom filters. *IEEE Transactions on Computers*.

[7] AlSabah, M. & Goldberg, I. (2016). Performance and security improvements for tor: A survey. *ACM Computing Surveys*, 49.

[8] Andersen, D., Balakrishnan, H., Kaashoek, F., & Morris, R. (2001). Resilient overlay networks. In *ACM Symposium on Operating Systems Principles*, SOSP '01 (pp. 131–145).

[9] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., & Zhou, Y. (2017). Understanding the Mirai botnet. In *USENIX Security Symposium*.

[10] Apostolaki, M., Maire, C., & Vanbever, L. (2021a). Perimeter: A network-layer attack on the anonymity of cryptocurrencies. In *Financial Cryptography and Data*

Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25 (pp. 147–166).: Springer.

[11] Apostolaki, M., Marti, G., Müller, J., & Vanbever, L. (2019). SABRE: Protecting Bitcoin against Routing Attacks. In *Network and Distributed System Security Symposium* (*NDSS*).

[12] Apostolaki, M., Singla, A., & Vanbever, L. (2021b). Performance-driven internet path selection. In *ACM SIGCOMM Symposium on SDN Research* (*SOSR*) (pp. 41–53).

[13] Apostolaki, M., Zohar, A., & Vanbever, L. (2017). Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *IEEE Symposium on Security and Privacy* (*S&P*).

[14] Arora, A. & Garman, C. (2025). Improving the Performance and Security of Tor's Onion Services. *Proceedings on Privacy Enhancing Technologies*.

[15] Aruba (2023). HPE greenlake for aruba (NaaS). https://www.arubanetworks.com/solutions/naas/.

[16] Aryaka (2026). Managed SD-WAN Solutions for the Cloud Era. https://www.aryaka.com/managed-wan-services/. Accessed: 2026-03-10.

[17] Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., & Teixeira, R. (2006). Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06 (pp. 153–158). New York, NY, USA: Association for Computing Machinery.

[18] Aumasson, J.-P. (2021). Siphash reference implementation. https://github.com/veorq/SipHash. Accessed: 2021-05-13.

[19] Aumasson, J.-P. & Bernstein, D. J. (2012). SipHash: A Fast Short-Input PRF. *Lecture Notes in Computer Science*, 7668.

[20] Avramopoulos, I. & Rexford, J. (2006). Stealth probing: Efficient Data-Plane security for IP routing. In *USENIX Annual Technical Conference* Boston, MA: USENIX Association.

[21] Bao, G. & Guo, P. (2022). Federated learning in cloud-edge collaborative architecture: key technologies, applications and challenges. *Journal of Cloud Computing*, 11.

[22] Beaulieu, R., Treatman-Clark, S., Shors, D., Weeks, B., Smith, J., & Wingers, L. (2015). The simon and speck lightweight block ciphers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference* (*DAC*) (pp. 1–6).

[23] Bertin, G. (2017). XDP in practice: Integrating XDP into our DDoS mitigation pipeline. In *Netdev: The Technical Conference on Linux Networking*.

[24] Birge-Lee, H., Sun, Y., Edmundson, A., Rexford, J., & Mittal, P. (2018). Bamboozling Certificate Authorities with BGP. In *USENIX Security Symposium*.

[25] Birge-Lee, H., Wang, L., McCarney, D., Shoemaker, R., Rexford, J., & Mittal, P. (2021). Experiences deploying Multi-Vantage-Point domain validation at let' s encrypt. In *30th USENIX Security Symposium* (*USENIX Security 21*) (pp. 4311–4327).: USENIX Association.

[26] Birge-Lee, H., Wang, L., Rexford, J., & Mittal, P. (2019). SICO: Surgical Interception Attacks by Manipulating BGP Communities. In *ACM SIGSAC Conference on Computer and Communications Security* (*CCS*).

[27] Birge-Lee, H., Yoo, S., Herber, B., Rexford, J., & Apostolaki, M. (2024a). TANGO: Secure collaborative route control across the public internet. In *21st USENIX Symposium on Networked Systems Design and Implementation* (*NSDI 24*) (pp. 1791–1811). Santa Clara, CA: USENIX Association.

[28] Birge-Lee, H., Yoo, S., Herber, B., Rexford, J., & Apostolaki, M. (2024b). TANGO: Secure collaborative route control across the public internet. In *21st USENIX Symposium on Networked Systems Design and Implementation* (*NSDI 24*): USENIX Association.

[29] Bischoff, P. (2025). Internet censorship: A map of internet censorship and restrictions. https://www.comparitech.com/blog/vpn-privacy/internet-censorship-map/.

[30] Bmv2 authors (2019). Behavioral model (bmv2). https://github.com/p4lang/behavioral-model. Accessed: 2021-09-28.

[31] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., & Walker, D. (2014). P4: Programming protocol-independent packet processors. In *ACM SIGCOMM Computer Communication Review*.

[32] Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., & Horowitz, M. (2013). Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM*.

[33] Broadcom (2023). BCM56870 Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series. Accessed: 2023-05-28.

[34] Cai, X., Nithyanand, R., Wang, T., Johnson, R., & Goldberg, I. (2014). A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security* (pp. 227–238).

[35] CAIDA (2018a). The CAIDA UCSD Anonymized Internet Traces 2018 - July 19th, equinix-nyc.dirA.20180719-130000.

[36] CAIDA (2018b). Trace Statistics for CAIDA Passive OC48 and OC192 Traces.

[37] CAIDA (2023). The CAIDA AS relationships dataset. https://www.caida.org/catalog/datasets/as-relationships/.

[38] Camacho, J. M., García-Martínez, A., Bagnulo, M., & Valera, F. (2013). BGP-XM: BGP Extended Multipath for Transit Autonomous Systems. *Computer Networks*, 57(4), 954–975.

[39] Chen, A., Zhou, W., Sriraman, A., Vaidya, T., Zhang, Y., Haeberlen, A., Loo, B., Phan, L., Sherr, M., & Shields, C. (2016). Dispersing Asymmetric DDoS Attacks with SplitStack. In *ACM Workshop on Hot Topics in Networks* (*HotNets*).

[40] Chen, X. (2020a). Implementing AES Encryption on Programmable Switches via Scrambled Lookup Tables. In *ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure* (*SPIN*).

[41] Chen, X. (2020b). Implementing AES encryption on programmable switches via scrambled lookup tables. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, SPIN '20 (pp. 8–14). New York, NY, USA: Association for Computing Machinery.

[42] Chen, X., Kim, H., Aman, J. M., Chang, W., Lee, M., & Rexford, J. (2020). Measuring tcp round-trip time in the data plane. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, SPIN '20 (pp. 35–41). New York, NY, USA: Association for Computing Machinery.

[43] Chen, X., Liu, H., Zhang, D., Huang, Q., Zhou, H., Wu, C., & Yang, Q. (2022). Empowering ddos attack mitigation with programmable switches. *IEEE Network*.

[44] Chen, X., Wu, C., Liu, X., Huang, Q., Zhang, D., Zhou, H., Yang, Q., & Khan, M. K. (2023). Empowering network security with programmable switches: A comprehensive survey. *IEEE Communications Surveys & Tutorials*.

[45] Cherubin, G., Jansen, R., & Troncoso, C. (2022). Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. In *31st USENIX Security Symposium* (*USENIX Security 22*) (pp. 753–770). Boston, MA: USENIX Association.

[46] Cilium Authors (2017). Cilium: ebpf-based networking, observability, security. https://cilium.io/. Accessed: 2022-12-01.

[47] Cimpanu, C. (2020). AWS said it mitigated a 2.3 Tbps DDoS attack, the largest ever. https://www.zdnet.com/article/aws-said-it-mitigated-a-2-3-tbps-ddos-attack-the-largest-ever/.

[48] Cisco (2017). Security Configuration Guide: Zone-Based Policy Firewall, Cisco IOS XE Fuji 16.7.x. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_data_zbf/configuration/xe-16-7/sec-data-zbf-xe-16-7-book/sec-ddos-attack-prevn.html. Accessed: 2023-09-15.

[49] Coker, J. (2022). Finland Government Sites Forced Offline by DDOS Attacks. https://www.infosecurity-magazine.com/news/finland-government-sites-offline/.

[50] Coleman, V. & Napolitano, J. (2022). Digital human rights need a single home in U.S. government. https://foreignpolicy.com/2022/03/14/digital-authoritarianism-tech-human-rights/.

[51] Collier, K., Dong, S., & Arouzi, A. (2022). Hacktivists, new and veteran, target Russia with one of cyber's oldest tools. https://www.nbcnews.com/tech/security/hacktivists-new-veteran-target-russia-one-cybers-oldest-tools-rcna20652.

[52] Consulting, I. T. I. (2019). Hourly Downtime Costs Rise. https://itic-corp.com/blog/2019/05/.

[53] De la Cadena, W., Mitseva, A., Hiller, J., Pennekamp, J., Reuter, S., Filter, J., Engel, T., Wehrle, K., & Panchenko, A. (2020). Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, (pp. 1971–1985).

[54] Decker, C. & Wattenhofer, R. (2013). Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings* (pp. 1–10).: IEEE.

[55] Demoulin, H. M., Pedisich, I., Vasilakis, N., Liu, V., Loo, B. T., & Phan, L. T. X. (2019). Detecting asymmetric application-layer Denial-of-Service attacks In-Flight with FineLame. In *2019 USENIX Annual Technical Conference* (*USENIX ATC 19*): USENIX Association.

[56] Deng, X., Li, Q., & Xu, K. (2024). Robust and reliable early-stage website fingerprinting attacks via spatial-temporal distribution analysis. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (pp. 1997–2011).

[57] Dimolianis, M., Pavlidis, A., & Maglaris, V. (2021). Syn flood attack detection and mitigation using machine learning traffic classification and programmable data plane filtering. In *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops* (*ICIN*) (pp. 126–133).

[58] Fayaz, S. K., Tobioka, Y., Sekar, V., & Bailey, M. (2015). Bohatei: Flexible and Elastic DDoS Defense. In *USENIX Security Symposium*.

[59] Feiner, L. (2022). Cyberattack hits Ukrainian banks and government websites. https://www.cnbc.com/2022/02/23/cyberattack-hits-ukrainian-banks-and-government-websites.html.

[60] Fichera, S., Galluccio, L., Grancagnolo, S. C., Morabito, G., & Palazzo, S. (2015). OPERETTA: An OPEnflow-based REmedy to mitigate TCP SYNFLOOD Attacks against Web Servers. *The International Journal of Computer and Telecommunications Networking*.

[61] Foundation, L. (2015). Data plane development kit (DPDK).

[62] Funk, A., Vesteinsson, K., & Baker, G. (2022). Countering an authoritarian overhaul of the internet. https://freedomhouse.org/report/freedom-net/2022/countering-authoritarian-overhaul-internet.

[63] Funk, A., Vesteinsson, K., & Baker, G. (2024). The struggle for trust online. https://freedomhouse.org/report/freedom-net/2024/struggle-trust-online.

[64] Gabriel (2020). What is Apache Keepalive Timeout? How to optimize this critical setting. https://ioflood.com/blog/2020/02/21/what-is-apache-keepalive-timeout-how-to-optimize-this-critical-setting/.

[65] Galov, N. (2022). 39 Jaw-Dropping DDoS Statistics to Keep in Mind for 2022. https://hostingtribunal.com/blog/ddos-statistics/#gref.

[66] Gao, L. & Rexford, J. (2001). Stable Internet Routing without Global Coordination. *IEEE/ACM Transactions on Networking*, 9(6), 681–692.

[67] Goldberg, S., Xiao, D., Tromer, E., Barak, B., & Rexford, J. (2008). Path-quality monitoring in the presence of adversaries. In *ACM SIGMETRICS* (pp. 193–204). New York, NY, USA: Association for Computing Machinery.

[68] Goldenberg, D. K., Qiu, L., Xie, H., Yang, Y. R., & Zhang, Y. (2004). Optimizing cost and performance for multihoming. In *ACM SIGCOMM* (pp. 79–92).: ACM.

[69] Goldschmidt, P. & Kučera, J. (2021). Defense Against SYN Flood DoS Attacks Using Network-based Mitigation Techniques. In *International Symposium on Integrated Network Management*: IEEE.

[70] Gomez, M. (2022). Dark Web Price Index 2020. https://www.privacyaffairs.com/dark-web-price-index-2020/.

[71] Gong, J. & Wang, T. (2020). Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX security symposium* (*USENIX security 20*) (pp. 717–734).

[72] Gran Alcoz, A., Strohmeier, M., Lenders, V., & Vanbever, L. (2022). Aggregate-based congestion control for pulse-wave ddos defense. In *ACM SIGCOMM*: ACM.

[73] Gutnikov, A., Kupreev, O., & Shmelev, Y. (2022a). DDoS Attacks in Q1 2022, Kapersky Lab Technical report. https://securelist.com/ddos-attacks-in-q1-2022. Accessed: 2022-04-15.

[74] Gutnikov, A., Kupreev, O., & Shmelev, Y. (2022b). DDoS Attacks in Q4 2021, Kapersky Lab Technical report. https://securelist.com/ddos-attacks-in-q4-2021. Accessed: 2022-04-15.

[75] Han, F., Wang, M., Cui, Y., Li, Q., Liang, R., Liu, Y., & Jiang, Y. (2022). Future Data Center Networking: From Low Latency to Deterministic Latency. *IEEE Network*, 36(1), 52–58.

[76] Haworth, J. (2022). Israeli government websites temporarily knocked offline by 'massive' cyber-attack. https://portswigger.net/daily-swig/israeli-government-websites-temporarily-knocked-offline-by-massive-cyber-attack.

[77] Hayes, J. & Danezis, G. (2016). k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium* (*USENIX Security 16*) (pp. 1187–1203).

[78] Henri, S., Garcia-Aviles, G., Serrano, P., Banchs, A., & Thiran, P. (2020). Protecting against website fingerprinting with multihoming. *Proceedings on Privacy Enhancing Technologies*.

[79] Holterbach, T., Molero, E. C., Apostolaki, M., Dainotti, A., Vissicchio, S., & Vanbever, L. (2019). Blink: Fast connectivity recovery entirely in the data plane. In *16th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI 19*) (pp. 161–176). Boston, MA: USENIX Association.

[80] Hummel, R., Hildebrand, C., Modi, H., Sockrider, G., Dobbins, R., Bjarnason, S., Sopko, J., DeSouza, S., Bondar, I., & Daff, O. (2019). *NETSCOUT Threat Intelligence Report for 2H 2019*. Technical report, Arbor Networks.

[81] Intel (2016). Barefoot Tofino. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html. "Accessed: 2023-09-20".

[82] Iselt, A., Kirstädter, A., Pardigon, A., & Schwabe, T. (2004). Resilient Routing Using MPLS and ECMP. In *IEEE Xplore*.

[83] Jain, S. (2021). What's in a name? Understanding the Google Cloud network "edge". https://cloud.google.com/blog/products/networking/understanding-google-cloud-network-edge-points.

[84] James, N. (2023). 45 Global DDOS Attack Statistics 2023. https://www.getastra.com/blog/security-audit/ddos-attack-statistics/. Accessed: 2023-01-17.

[85] Juarez, M., Imani, M., Perry, M., Diaz, C., & Wright, M. (2016). Toward an efficient website fingerprinting defense. In *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21* (pp. 27–46).: Springer.

[86] Juen, J., Johnson, A., Das, A., Borisov, N., & Caesar, M. (2015). Defending Tor from Network Adversaries: A Case Study of Network Path Prediction. *Proceedings on Privacy Enhancing Technologies*, 2015(2), 171–187.

[87] Juniper (2022). Juniper Networks' MX480 Universal Routing Platform. https://www.juniper.net/us/en/products/routers/mx-series/mx480-universal-routing-platform.html. Accessed: 2023-05-28.

[88] Kandula, S., Katabi, D., Davie, B., & Charny, A. (2005). Walking the tightrope: Responsive yet stable traffic engineering. In *ACM SIGCOMM*, SIGCOMM '05 (pp. 253–264). New York, NY, USA: Association for Computing Machinery.

[89] Kashaf, A., Sekar, V., & Agarwal, Y. (2020). Analyzing third party service dependencies in modern web services: Have we learned from the mirai-dyn incident? In *Proceedings of the ACM Internet Measurement Conference* (pp. 634–647).

[90] Kennedy, P. (2020). Intel Tofino2 Next-Gen Programmable Switch Detailed. https://www.servethehome.com/intel-tofino2-next-gen-programmable-switch-detailed/. Accessed: 2023-05-28.

[91] Khan, M. A., Baccour, E., Chkirbene, Z., Erbad, A., Hamila, R., Hamdi, M., & Gabbouj, M. (2022). A Survey on Mobile Edge Computing for Video Streaming: Opportunities and Challenges. *IEEE Access*, 10, 120514–120550.

[92] Kim, C., Sivaraman, A., Katta, N. P., Bas, A., Dixit, A., & Wobker, L. J. (2015). In-band network telemetry via programmable dataplanes. In *Proceedings of the ACM SIGCOMM 2015 Conference*. Industrial demo.

[93] Kim, D., Nelson, J., Ports, D. R. K., Sekar, V., & Seshan, S. (2021). RedPlane: Enabling Fault-Tolerant Stateful In-Switch Applications. In *ACM SIGCOMM*.

[94] Kim, D., Zhu, Y., Kim, C., Lee, J., & Liu, S. S. (2018). Generic External Memory for Switch Data Planes. In *ACM Workshop on Hot Topics in Networks* (*HotNets*).

[95] Koch, T., Yu, S., Agarwal, S., Katz-Bassett, E., & Beckett, R. (2023). Painter: Ingress traffic engineering and routing for enterprise cloud networks. In *Proceedings of the*

ACM SIGCOMM 2023 Conference, ACM SIGCOMM '23 (pp. 360–377). New York, NY, USA: Association for Computing Machinery.

[96] Kupreev, O., Badovskaya, E., & Gutnikov, A. (2020). DDoS Attacks in Q2 2020, Kapersky Lab Technical report. https://securelist.com/ddos-attacks-in-q2-2020. Accessed: 2022-04-15.

[97] Kupreev, O., Gutnikov, A., & Shmelev, Y. (2022). DDoS Attacks in Q3 2022, Kapersky Lab Technical report. https://securelist.com/ddos-report-q3-2022/107860/. Accessed: 2023-01-25.

[98] Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C. B., Shi, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., Bailey, J., Dorfman, J. C., Roskind, J., Kulik, J., Westin, P. G., Tenneti, R., Shade, R., Hamilton, R., Vasiliev, V., & Chang, W.-T. (2017). The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*.

[99] Lee, M., Goldberg, S., Kompella, R. R., & Varghese, G. (2014). Finecomb: Measuring microscopic latency and loss in the presence of reordering. *IEEE/ACM Transactions on Networking*, 22(4), 1136–1149.

[100] Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3), 50–60.

[101] Li, Y., Miao, R., Kim, C., & Yu, M. (2016). FlowRadar: A better NetFlow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (pp. 311–324). Santa Clara, CA: USENIX Association.

[102] Liebhaber, L. (2022). Countries with the most—and least—digital freedom. https://www.beyondidentity.com/resource/countries-with-the-most--and-least--digital-freedom.

[103] Linux kernel source (2021). Linux SYN cookie epoch time. https://elixir.bootlin.com/linux/v6.0/source/include/net/tcp.h#L497. Accessed: 2021-09-28.

[104] Linux man-pages project (2005). tcp(7) - Linux man page. https://linux.die.net/man/7/tcp. Accessed: 2021-09-28.

[105] Liu, Z., Namkung, H., Nikolaidis, G., Lee, J., Kim, C., Jin, X., Braverman, V., Yu, M., & Sekar, V. (2021). Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches. In *USENIX Security Symposium*.

[106] Mathews, N., Holland, J. K., Hopper, N., & Wright, M. (2024). Laserbeak: Evolving website fingerprinting attacks with attention and multi-channel feature representation. *IEEE Transactions on Information Forensics and Security*, 19, 9285–9300.

[107] Meier, R., Holterbach, T., Keck, S., Stähli, M., Lenders, V., Singla, A., & Vanbever, L. (2019). (self) driving under the influence: Intoxicating adversarial network inputs. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (pp. 34–42).

[108] Miao, R., Zeng, H., Kim, C., Lee, J., & Yu, M. (2017). SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM SIGCOMM*.

[109] Mogul, J. & Deering, S. (1990). *Path MTU discovery*. RFC 1191, RFC Editor. http://www.rfc-editor.org/rfc/rfc1191.txt.

[110] Mohammadi, R., Javidan, R., & Mauro, C. (2017). SLICOTS: An SDN-Based Lightweight Countermeasure for TCP SYN Flooding Attacks. *IEEE Transactions on Network and Service Management*, 14.

[111] Moura, G. C. M., Heidemann, J., Hardaker, W., Charnsethikul, P., Bulten, J., Ceron, J. M., & Hesselman, C. (2022). Old but gold: Prospecting TCP to engineer and live monitor DNS anycast. In *Proceedings of the Passive and Active Measurement Workshop* (pp. to appear). virtual: Springer.

[112] Narayana, S., Sivaraman, A., Nathan, V., Goyal, P., Arun, V., Alizadeh, M., Jeyakumar, V., & Kim, C. (2017). Language-directed hardware design for network performance monitoring. In *ACM SIGCOMM*, SIGCOMM '17 (pp. 85–98). New York, NY, USA: Association for Computing Machinery.

[113] Nichols, K. (2017). pping (pollere passive ping).

[114] Nvidia (2022). Nvidia BlueField-3 DPU Programmable Data Center Infrastructure On-a-Chip. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf. Accessed: 2023-05-28.

[115] Osama, M., Ateya, A. A., Ahmed Elsaid, S., & Muthanna, A. (2022). Ultra-Reliable Low-Latency Communications: Unmanned Aerial Vehicles Assisted Systems. *Advances in Wireless Communications Systems, Information*, 13.

[116] Ostermann, S. (2007). tcptrace homepage.

[117] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., & Wehrle, K. (2016). Website fingerprinting at internet scale. In *NDSS*, volume 1 (pp. 23477).

[118] Panchenko, A., Niessen, L., Zinnen, A., & Engel, T. (2011). Website Fingerprinting in Onion Routing based Anonymization Networks. In *ACM Workshop on Privacy in the Electronic Society* (*WPES*).

[119] Paxson, V. (1997). End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5.

[120] Penkov, P., Dumazet, E., & Fomichev, S. (2020). Issuing SYN Cookies in XDP. In *Netdev, The Technical Conference on Linux Networking*.

[121] Perrig, A., Szalachowski, P., Reischuk, R. M., & Chuat, L. (2017). *SCION: A Secure Internet Architecture*. Springer Verlag.

[122] Pilosov, A. & Kapela, T. (2008). Stealing the Internet: An Internet-scale Man in the Middle Attack. *NANOG 44*.

[123] Pinho, M. (2021). AWS Shield threat landscape review: 2020 year-in-review. https://aws.amazon.com/blogs/security/aws-shield-threat-landscape-review-2020-year-in-review/.

[124] Qiu, T., Ni, J., Wang, H., Hua, N., Yang, Y. R., & Xu, J. J. (2008). Packet doppler: Network monitoring using packet shift detection. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08 New York, NY, USA: Association for Computing Machinery.

[125] Rahouti, M., Xiong, K., Ghani, N., & Shaikh, F. (2020). SYNGuard: Dynamic threshold-based SYN flood attack detection and mitigation in software-defined networks. *The Institution of Engineering and Technology Networks*.

[126] Rekhter, Y., Li, T., & Hares, S. (2006). *A Border Gateway Protocol 4* (*BGP-4*). RFC 4271, RFC Editor. http://www.rfc-editor.org/rfc/rfc4271.txt.

[127] Rosson, Z., Felicia, & Tackett, C. (2024). The most violent year: internet shutdowns in 2023. https://www.accessnow.org/internet-shutdowns-2023/.

[128] Saad, M., Cook, V., Nguyen, L., Thai, M. T., & Mohaisen, A. (2019). Partitioning attacks on bitcoin: Colliding space, time, and logic. In *2019 IEEE 39th international conference on distributed computing systems* (*ICDCS*) (pp. 1175–1187).: IEEE.

[129] Schlinker, B., Kim, H., Cui, T., Katz-Bassett, E., Madhyastha, H. V., Cunha, I., Quinn, J., Hasan, S., Lapukhov, P., & Zeng, H. (2017). Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *ACM SIGCOMM*.

[130] Scholz, D., Gallenmuller, S., Stubbe, H., Jaber, B., Rouhi, M., & Carle, G. (2020). Me Love (SYN-)Cookies: SYN Flood Mitigation in Programmable Data Planes. In *P4 Workshop in Europe* (*EUROP4*): Open Networking Foundation.

[131] Scholz, D., Oeldemann, A., Geyer, F., Gallenmuller, S., Stubbe, H., Wild, T., Herkersdorf, A., & Carle, G. (2019). Cryptographic Hashing in P4 Data Planes. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*.

[132] Security, C. (2023). TTL Expiry Attack Identification and Mitigation . https://sec.cloudapps.cisco.com/security/center/resources/ttl_expiry_attack.html#2. Accessed: 2023-05-17.

[133] Sengupta, S., Kim, H., & Rexford, J. (2022). Continuous in-network round-trip time monitoring. In *ACM SIGCOMM*, SIGCOMM '22 (pp. 473–485). New York, NY, USA: Association for Computing Machinery.

[134] Services, A. W. (2022). *AWS Best Practices for DDoS Resiliency*. Technical report, AWS Whitepaper.

[135] Severi, G., Jagielski, M., Yar, G., Wang, Y., Oprea, A., & Nita-Rotaru, C. (2022). Network-level adversaries in federated learning. In *2022 IEEE Conference on Communications and Network Security* (*CNS*) (pp. 19–27).: IEEE.

[136] Shin, S., Yegneswaran, V., Porras, P., & Gu, G. (2013). AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks. In *ACM SIGSAC Conference on Computer & Communications Security*.

[137] Shirokov, N. & Dasineni, R. (2018). Open-sourcing Katran, a scalable network load balancer. https://engineering.fb.com/2018/05/22/open-source/open-sourcing-katran-a-scalable-network-load-balancer/.

[138] Shukla, A. & Foerster, K.-T. (2022). Shortcutting fast failover routes in the data plane. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ANCS '21 (pp. 15–22). New York, NY, USA: Association for Computing Machinery.

[139] Simsek, G., Ergenç, D., & Onur, E. (2021). Efficient network monitoring via in-band telemetry. In *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)* (pp. 1–6).

[140] Sirinam, P., Imani, M., Juarez, M., & Wright, M. (2018). Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1928–1943).

[141] Sirinam, P., Mathews, N., Rahman, M. S., & Wright, M. (2019). Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1131–1148).

[142] Smith, J.-P., Dolfi, L., Mittal, P., & Perrig, A. (2022). {QCSD}: A {QUIC}{Client-Side}{Website-Fingerprinting} defence framework. In *31st USENIX Security Symposium (USENIX Security 22)* (pp. 771–789).

[143] Snader, R. & Borisov, N. (2011). Improving security and performance in the tor network through tunable path selection. *IEEE Trans. Dependable Sec. Comput.*, 8, 728–741.

[144] Snijders, J. (2016). Practical everyday BGP filtering with AS_PATH filters: Peer locking. *NANOG-67*.

[145] Sonchack, J., Loehr, D., Rexford, J., & Walker, D. (2021). Lucid: a Language for Control in the Data Plane. In *ACM SIGCOMM*: ACM.

[146] Streibelt, F., Lichtblau, F., Beverly, R., Feldmann, A., Pelsser, C., Smaragdakis, G., & Bush, R. (2018a). Bgp communities: Even more worms in the routing can. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18 (pp. 279–292). New York, NY, USA: Association for Computing Machinery.

[147] Streibelt, F., Lichtblau, F., Beverly, R., Feldmann, A., Pelsser, C., Smaragdakis, G., & Bush, R. (2018b). Bgp communities: Even more worms in the routing can. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18 (pp. 279–292). New York, NY, USA: Association for Computing Machinery.

[148] Sun, G., Jiang, M., Khooi, X. Z., Li, Y., & Li, J. (2023). Neobft: Accelerating byzantine fault tolerance using authenticated in-network ordering. In *ACM SIGCOMM* (pp. 239–254).

[149] Sun, Y., Edmundson, A., Feamster, N., Chiang, M., & Mittal, P. (2017). Counter-RAPTOR: Safeguarding Tor Against Active Routing Attacks. In *2017 IEEE Symposium on Security and Privacy* (*SP*): IEEE.

[150] Sun, Y., Edmundson, A., Vanbever, L., Li, O., Rexford, J., Chiang, M., & Mittal, P. (2015a). RAPTOR: Routing Attacks on Privacy in Tor. In *USENIX Security Symposium*.

[151] Sun, Y., Edmundson, A., Vanbever, L., Li, O., Rexford, J., Chiang, M., & Mittal, P. (2015b). Raptor: routing attacks on privacy in tor. In *Proceedings of the 24th USENIX Conference on Security Symposium* USA: USENIX Association.

[152] Suricata authors (2018). Suricata - eBPF and XDP. https://suricata.readthedocs.io/en/latest/capture-hardware/ebpf-xdp.html.

[153] Tang, W., Kiffer, L., Fanti, G., & Juels, A. (2023). Strategic latency reduction in blockchain peer-to-peer networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 7(2), 1–33.

[154] TechNet, M. (2014). Syn attack protection on Windows Vista, Windows 2008, Windows 7, Windows 2008 R2, Windows 8/8.1, Windows 2012 and Windows 2012 R2. https://docs.microsoft.com/en-us/answers/questions/144446/synattackprotect.html. Accessed: 2022-01-28.

[155] Technologies, C. P. S. (2021). Understanding Aggressive Aging. https://sc1.checkpoint.com/documents/R80.20/SmartConsole_OLH/EN/html_frameset.htm?topic=documents/R80.20/SmartConsole_OLH/EN/Wh_4163Q-r2uASm5pwt7Iw2. Accessed: 2023-09-15.

[156] Techy News Today (2024). Guardians of your data: Exploring privacy enhancing technology. https://techynewstoday.com/guardians-of-your-data-exploring-privacy-enhancing-technology/.

[157] The FreeBSD Project (2008). FreeBSD manual page for syncookies. https://www.freebsd.org/cgi/man.cgi?syncookies. Accessed: 2022-10-01.

[158] Valancius, V., Ravi, B., Feamster, N., & Snoeren, A. C. (2013). Quantifying the benefits of joint content and network routing. *SIGMETRICS Perform. Eval. Rev.*, 41(1), 243–254.

[159] Versa Networks (2026). Cloud WAN. https://versa-networks.com/sd-wan/cloud-wan/. Accessed: 2026-03-10.

[160] Vultr (2022a). Announce your IP space with BGP and Vultr - Vultr.com. https://www.vultr.com/features/bgp/.

[161] Vultr (2022b). AS20473 BGP customer guide. https://www.vultr.com/docs/as20473-bgp-customer-guide.

[162] Vultr (2022c). SSD VPS servers, cloud servers and cloud hosting. https://www.vultr.com/.

[163] Wang, L., Kim, H., Mittal, P., & Rexford, J. (2020a). Programmable in-network obfuscation of traffic. *CoRR*, abs/2006.00097.

[164] Wang, L., Kim, H., Mittal, P., & Rexford, J. (2020b). Programmable in-network obfuscation of traffic. *CoRR*, abs/2006.00097.

[165] Wang, L., Kim, H., Mittal, P., & Rexford, J. (2023). Raven: Stateless rapid ip address variation for enterprise networks. *Proceedings on Privacy Enhancing Technologies*, 2023.

[166] Wang, L., Mittal, P., & Rexford, J. (2022a). Data-plane security applications in adversarial settings. In *ACM SIGCOMM Computer Communication Review*.

[167] Wang, M., Kulshrestha, A., Wang, L., & Mittal, P. (2022b). Leveraging strategic connection migration-powered traffic splitting for privacy. *Proceedings on Privacy Enhancing Technologies*, 2022, 498–515.

[168] Wang, T., Cai, X., Nithyanand, R., Johnson, R., & Goldberg, I. (2014). Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium* (*USENIX Security 14*) (pp. 143–157).

[169] Wirz, F., Gartner, M., van Bommel, J., Moghadam, E. E., Cimaszewski, G. H., He, A., Zhang, Y., Birge-Lee, H., Kottmann, F., Krähenbühl, C., Kwon, J., Mavromati, K., Wang, L., Bertolo, D., Canini, M., Cho, B., Ferreira, R. A., Green, S. P., Hausheer, D., Hur, J., Jia, X., Lee, H., Mittal, P., Oaiya, O., Park, C., Perrig, A., Sobieski, J., Sun, Y., Wang, C., & Wierenga, K. (2025). SIGCOMM '25 Results for Scaling SCIERA: A Journey Through the Deployment of a Next-Generation Network. Dataset and artifact repository. Accessed 2026-05-06.

[170] Yang, Z., Wu, Z., Luo, M., Chiang, W.-L., Bhardwaj, R., Kwon, W., Zhuang, S., Luan, F. S., Mittal, G., Shenker, S., et al. (2023). {SkyPilot}: An intercloud broker for sky computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (pp. 437–455).

[171] Yap, K.-K., Motiwala, M., Rahe, J., Padgett, S., Holliman, M., Baldus, G., Hines, M., Kim, T., Narayanan, A., Jain, A., Lin, V., Rice, C., Rogan, B., Singh, A., Tanaka, B., Verma, M., Sood, P., Tariq, M., Tierney, M., Trumic, D., Valancius, V., Ying, C., Kallahalla, M., Koley, B., & Vahdat, A. (2017). Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of ACM SIGCOMM 2017* (pp. 432–445). New York, NY, USA: Association for Computing Machinery.

[172] Yoo, S. & Chen, X. (2021). Secure Keyed Hashing on Programmable Switches. In *ACM SIGCOMM Workshop on Secure Programmable network Infrastructure (SPIN'21)*: ACM.

[173] Yoo, S., Chen, X., & Rexford, J. (2024). SMARTCOOKIE: blocking large-scale SYN floods with a split-proxy defense on programmable data planes. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24: USENIX Association.

[174] Yoo, S., Sengupta, S., Apostolaki, M., & Rexford, J. (2023). Sieve: Layered network defenses against large-scale attacks. In *P4 Workshop*.

[175] Yoshinaka, Y., Takemasa, J., Koizumi, Y., & Hasegawa, T. (2022). On implementing chacha on a programmable switch. In *Proceedings of the 5th International Workshop on P4 in Europe*, EuroP4 '22 (pp. 15–18). New York, NY, USA: Association for Computing Machinery.

[176] Yuliang, L., Rui, M., Changhoon, K., & Minlan, Y. (2016). LossRadar: Fast Detection of Lost Packets in Data Center Networks. In *CoNEXT* New York, NY, USA: ACM.

[177] Zhang, M., Li, G., Wang, S., Liu, C., Chen, A., Hu, H., Gu, G., Li, Q., Xu, M., & Wu, J. (2020). Poseidon: Mitigating volumetric DDoS attacks with programmable switches. In *Network and Distributed System Security Symposium*.

[178] Zhang, X., Chen, H., Zhao, Y., Ma, Z., Xu, Y., Huang, H., Yin, H., & Wu, D. O. (2019). Improving Cloud Gaming Experience through Mobile Edge Computing. *IEEE Wireless Communications*, 26(4), 178–183.

[179] Zhang, X., Sen, T., Zhang, Z., April, T., Chandrasekaran, B., Choffnes, D., Maggs, B. M., Shen, H., Sitaraman, R. K., & Yang, X. (2021). Anyopt: Predicting and optimizing ip anycast performance. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21 (pp. 447–462). New York, NY, USA: Association for Computing Machinery.

[180] Zhang, Z., Zhang, M., Greenberg, A., Hu, Y. C., Mahajan, R., & Christian, B. (2010). Optimizing cost and performance in online service provider networks. In *USENIX Networked Systems Design and Implementation*.

[181] Zhou, H., Hong, S., Liu, Y., Luo, X., Li, W., & Gu, G. (2023). Mew: Enabling large-scale and dynamic link-flooding defenses on programmable switches. In *2023 IEEE Symposium on Security and Privacy* (*SP*).