Infrastructure-based Anonymous Communication Protocols in Future Internet Architectures

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Chen Chen

B.S., Department of Automation, Tsinghua University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

August 2018

Acknowledgments

I would like to thank every person who helped me throughout my Ph.D. journey.

First and foremost, I would like to express my deepest gratefulness to my dearest advisor, Dr. Adrian Perrig, for his mentoring, insights, and inspiration. His dedication and enthusiasm for research always motivate me to dive into new research areas and face new challenges. His guidance on both my academic and everyday life navigates me through difficult times. His insights and constructive criticism inspire me to conquer doubts and live up to my full potential.

I would also like to thank my thesis committee members for all their assistance and insightful feedback: Dr. Lujo Bauer, Dr. George Danezis, Dr. Vyas Sekar. I'm also deeply indebted to all my collaborators: Daniele E. Asoni, Dr. Fan Bai, Dr. David Barrera, Dr Hsu-Chun Hsiao, Dr. Petros Maniatis, Dr. Steve Matsumoto, Dr. Himanshu Raj, Dr. Stefan Saroiu, Dr. Ahren Studer, Dr. Carmela Troncoso, Dr. Amit Vasudevan, Dr. Alec Wolman, Dr. James Zeng. I'm extremely fortunate to have worked with so many talented and hardworking researchers. I'm particularly grateful to Dr. Stefan Saroiu, my mentor at Microsoft research, who taught me about essential skills on critical thinking, which greatly benefits my journey.

I also greatly appreciate the support from the faculty members and staffs at both Carnegie Mellon University and ETH Zürich. Specifically, I would like to thank Dr. David Brumley, Dr. Vigil Gligor, Katharina Schuppli, Stephanie Scott, and Sari Smith, who have helped me in research and life.

I could not complete the journey without support from my parents and accompany of my caring friends. Thanks to my friends for sharing ideas, beers, happiness, and sorrow: Dr. Jun Han, Dr. Taeho Lee, Wei Li, Dr. Yanlin Li, Yinong Lin, Dr. Yue-Hsun Lin, Dr. Chris Pappas, Dr. Zongwei Zhou, Dr. Xiao Wang, Miao Yu, Tianlong Yu, and Dr. Yao Zhang. I would also specially thank Dr. Yuan Tian for always trusting me and supporting me. Thanks to my parents for unconditionally believing in me and supporting me all along.

Lastly, the research towards this degree is funded by NSF under the grant number CNS-1040801, European Union's Seventh Framework Programes (FP7/2007-2013) / ERC grant agreement 617605, and the Intel Science and Technology Center for Secure Computing. I gratefully acknowledge all the support from these sources, without which it is impossible to write this dissertation.

Abstract

User anonymity faces increasing threats from private companies, network service providers, and governmental surveillance programs. Current anonymous communication systems running as overlay networks offer neither satisfactory performance to support diverse Internet applications nor strong security guarantees. As Future Internet Architectures emerge and propose to equip routers with cryptographic operations, this thesis aims to answer the question: what level of security guarantee and performance can anonymous communication system offer if designed as a service of the network infrastructure?

This thesis thus presents three scalable and highly efficient infrastructure-based anonymous communication systems, HORNET, PHI, and TARANET, defeating adversaries ranging from a single malicious Internet Service Provider to governments conducting mass surveillance. Our contributions are summarized below:

1. We present HORNET, a low-latency onion routing system that operates at the network layer thus enabling a wide range of applications. HORNET uses only symmetric cryptography for data forwarding and requires no per-flow state on intermediate routers to achieve high scalability. This design enables HORNET routers implemented on off-the-shelf workstation to process anonymous traffic at over 93 Gb/s.

2. We propose PHI, a Path-HIdden lightweight anonymity protocol that fixes two vulnerabilities of LAP and Dovetail. We present an efficient packet header format that hides path information and a new back-off setup method that is compatible with current and future network architectures. Our experiments demonstrate that PHI expands anonymity sets of LAP and Dovetail by over 30x and reaches 120 Gb/s forwarding speed on a commodity software router.

3. We propose TARANET, an anonymity system that implements protection against traffic analysis at the network layer, and limits the incurred latency and overhead. In TARANET's setup phase, traffic analysis is thwarted by mixing. In the data transmission phase, end hosts and Autonomous Systems coordinate to shape traffic into constant-rate transmission using packet splitting. Our prototype implementation shows that TARANET can forward anonymous traffic at over 50 Gb/s using commodity hardware.

In summary, this thesis demonstrates that it is not only viable but also beneficial to build infrastructure-based anonymous communication systems. The proposed schemes achieve a new level of scalability and performance and characterize a general trade-off between anonymity guarantees and performance that guides future infrastructure-based anonymous communication system designs.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The advent of the Internet age has brought us into a post-privacy world [168]. On the one hand, we have witnessed unprecedented ease of access to information. On the other hand, the availability of personal information and the massive data processing capability render users transparent. An increasing number of privacy-aware users respond to the undesirable transparency by actively seeking anonymity. In fact, a popular anonymity service, Tor [79], attracts 4 million daily users [25].

Whether Internet anonymity is a boon to society has been a controversial topic, as anonymity enables both legitimate and illegitimate use cases. The examples below summarize standard usage of today's anonymity services.

- Freedom of expression. People want separation between their online expression and offline life, as they may fear unfair judgment, harassment, or retribution. For instance, a user would like to post an answer to a sensitive question online without being judged by an intolerant community. A reviewer would like to write an honest review without being harassed by the business owner. A journalist or a whistleblower wants to report news while avoiding political or economic retribution.

- Personal information protection. People want to prevent their sensitive data from leaking to the public or being linked to their identities. For example, a smartphone user would like to conceal his/her geographic location. A patient wants to keep his/her medical record private. An online consumer would like to hide his/her browsing history from advertisement companies or spammers.

- Criminal activities. Criminals tend to exploit anonymity services to conceal their identity to avoid conviction and punishment. A terrorist can use an anonymous chat service to organize an attack. A hacker can collect ransoms from malware victims through an anonymous financial service. A drug dealer can anonymously sell illegal drugs anonymously.

Though sometimes misused to conduct crimes, anonymity, as an option and as a fundamental right, has a significant role in today's political and social discourse. A 1995 Supreme Court ruling in McIntyre v. Ohio Elections Commission reads:

> *Anonymity is a shield from the tyranny of the majority. . . . It thus exemplifies the purpose behind the Bill of Rights and of the First Amendment in particular: to protect unpopular individuals from retaliation . . . at the hand of an intolerant society.*

Moreover, anonymity is also valuable to privacy-insensitive people to avoid future unintended consequences. Because collected data can stay forever but the society keeps evolving, one's insensitive information today may become sensitive tomorrow. For example, a person who is associated with insensitive political statements under the current government can be subject to prosecution when another government rules.

In this thesis, we argue that anonymous communication should become a built-in service of the Internet to guarantee a person's right to free speech. Unfortunately, the current Internet only offers anonymity through optional application-layer services, which provides neither strong anonymity guarantees nor satisfactory performance.

1.1 Status of Internet Anonymity

Anonymity is not a feature of the current Internet by design. For example, a packet's sender IP address is usually available to the receiver, revealing the sender's network location. Web services transport unencrypted metadata (e.g., cookies) that identify users. Moreover, by granting easy access to users' personal information, the Internet incentivizes and empowers adversaries of different capabilities to track users for business or political purposes. For instance, web applications [45], cellphone vendors [119], and Internet Service Providers (ISP) [175] have all been found to track users secretly. Social networks collect personal information without users' awareness [90] and third-party applications misuse private information to influence elections [4]. Recent revelations also bring to light mass surveillance programs that covertly collect not only international but also domestic intelligence [9, 21]. We summarize the best known anonymity threats regarding tracking entities below.

- Application service providers. An application service provider has an incentive to track users' locations and behavior and sell the information to advertisers. An application service provider typically has access to users' traffic sent to and from the application.

- Internet service providers. An Internet service provider can also identify users through logging and analyzing unencrypted traffic. Compared to application providers, network service providers are able to observe all traffic of a user and thus profile a user's behavior across applications.

- Governments. A government can access large-scale computation and storage. It can also coerce multiple network service providers to cooperate and install on-site surveillance equipment.

In addition to tracking entities, tracking methods can also vary widely. We broadly group existing tracking methods that are either observed in the Internet or proposed by the research community into the following two categories.

- Using identifiers within network traffic. By intercepting and analyzing packets, a tracking entity can extract the information that identifies users, such as browser cookies. The tracking entity can then selectively promote targeted users' traffic for detailed analysis. Because extracting and matching in-packet identifiers is efficient, several tracking entities are known to use this method to track users [9,21,175].

- Traffic analysis. Traffic analysis attacks are attacks against user anonymity that leverage both packet contents and cross-packet metadata, such as statistical patterns and timing information. Depending on whether a traffic analysis attack causes detectable effects, we can further categorize traffic analysis attacks into *passive attacks* and *active attacks*. Passive attacks focus on analyzing observed traffic and metadata, and active attacks aggressively modify network traffic to trigger observable effects. The research community has demonstrated a range of passive and active traffic analysis attacks that defeat existing anonymity services [118,127,133,200]. However, compared to using identifiers within network traffic to track users, traffic analysis attacks are of higher computation cost and require the attacker's presence at multiple locations. It remains an open question whether traffic analysis attacks are practical for mass surveillance.

Facing such varied and resourceful adversaries, users only have access to a small set of application-layer services to retain anonymity. We list popular anonymity services below.

- Anonymous Virtual Private Network (VPN). Anonymous VPNs promise not to track users. Using an anonymous VPN, a user can hide his/her network location by routing all traffic through the VPN. However, the shortcoming of an anonymous VPN is straightforward: the user needs to trust the anonymous VPN to be benign. In case the anonymous VPN is malicious or compromised, the user's anonymity is lost.

- Onion routing network. An onion routing network (e.g., Tor [79] and I^2P [196]) is an application overlay composed of servers. A client anonymizes its messages by selecting a sequence of servers and routing each message through the appointed servers. The key feature of the onion routing network is to use multiple layers of encryption/decryption for protecting a message, where each layer corresponds to one traversed server. Messages are grouped into sessions to avoid renegotiating new cryptographic secrets within each session. Onion routing networks offer bandwidth and latency suitable for browsing web pages, and they are the most popular anonymity services. The level of anonymity offered by onion routing networks is insufficient because the onion routing networks are vulnerable to various traffic analysis attacks [118, 127, 133, 200].

- Mix network. A mix network adds protection against traffic analysis attacks to an onion routing network by obscuring packet timing information [131]. However, the protection mechanisms add such long delay for messages that real-time web browsing is no longer possible. Consequently, mix networks are typically suitable for delay-tolerant applications, such as email or messaging.

Unfortunately, all existing anonymity solutions have unsatisfactory performance due to their nature of overlay networks. First, because each hop within an overlay network can reroute a packet across the underlying network, the path length of an overlay network is large, which results in high propagation latency. Second, because an overlay node runs as an application, an anonymous packet additionally has to traverse the entire kernel stack, increasing processing latency. Third, in onion routing networks (e.g., Tor [79]) and mix networks, packets are queued in multiple layers of the network stack, causing high queuing delays. Finally, volunteered machines cannot achieve high throughput and performance as dedicated network routers [5].

1.2 Infrastructure-based Anonymous Communication: Benefits and Challenges

Recently, researchers propose clean-slate designs for Future Internet Architectures (FIA) offering desirable properties that the current Internet design lacks, such as isolation of failures, path control, trustworthiness, and evolvability. For instance, Named Data Networking regards content as the first-class principal, and expedites content distribution through in-network caching [197]. NEBULA offers policy-compliant paths and enables each router to verify the origins of the data carried [35]. Mobility First focuses on adapting to dynamic hosts and enables trustworthy, robust, and scalable network mobility [150]. XIA aims at providing a flexible and evolvable network architecture that integrates rich addressing of alternate first-class principals, such as content, services, or users [34]. SCION providers isolation, path control, scalability, and high availability [198].

Compared to network devices such as routers in the current Internet, network devices in FIAs are assumed to provide additional functionalities. For example, in Named Data Networking, routers are capable of caching contents [197]. In NEBULA [35] and SCION [198], a router needs to conduct cryptographic operations. In XIA, routers need to parse addresses in the format of a Directed Acyclic Graph (DAG) [34].

In this thesis, we consider a network architecture that provisions anonymous communication as a core service. In particular, we assume that network devices are capable of cryptographic operations during packet forwarding. Compared to existing anonymity systems based on overlay networks, an infrastructure-based anonymity network inherently features three advantages. First, shorter paths between the source and the destination (instead of the long paths inflated by redirection in overlay networks) can be used, reducing propagation latency. Second, only layers below the transport layer in the network stack are involved, eliminating processing and excessive queuing delay of upper layers in overlay networks. Third, routers can provide higher throughput than today's voluntarily-contributed servers, with the promise to increase the throughput of anonymous communication.

However, to design and build an infrastructure-based anonymity network, we face immediate challenges: scalability, the trade-off between security guarantees and performance, and intrinsic topology-based de-anonymization. First, current anonymity systems require additional state stored in relay nodes. The new state required to provide anonymity can cause scalability problems: excessive state could overload the high-speed memory in today's routers. Second, it remains unclear what level of anonymity guarantees an infrastructure-based anonymity network can provide to achieve high throughput of current commodity routers. For instance, an onion routing scheme with traffic mixing can resist traffic analysis, but using traffic mixing can introduce long processing latency and limit the routers' total throughput [116]. Finally, traffic among ISPs follows the ISPs' network policies, thus preventing global redirection used in overlay networks. Consequently, even receiving a single packet from a neighbor ISP allows an adversarial ISP to infer end hosts' network locations.

1.3 Thesis Statement

We argue that it is possible to design scalable and high-speed anonymous communication systems as an essential service of network architectures to satisfy a variety of anonymity and performance requirements. In this context, we propose the following thesis statement:

Based on a future Internet architecture, it is possible to design scalable anonymous communication systems that simultaneously achieve traffic-analysis resistance, scalability, and high performance on today's commercial routers. It is also possible to trade the level of anonymity for higher performance, yielding

more efficient designs that defeat weaker adversaries.

To validate our thesis, we propose three infrastructure-based anonymous communication systems that respectively resist a local adversary, a global passive adversary incapable of traffic analysis, and a global adversary capable of traffic analysis. The three systems exemplify a new trade-off between anonymity and performance for FIA-based anonymity system designs.

1.3.1 Desired Privacy Properties

The designed system should achieve the following privacy properties:

- Large anonymity set size. Anonymity can be defined as keeping the sender or receiver unidentifiable within a set of subjects called an *anonymity set* [144]. The proposed systems should achieve large anonymity sets to provide strong anonymity guarantees.
- Session unlinkability. An adversary cannot link packets from different sessions, even those between the same set of sources and destinations.
- Path information integrity and secrecy. An adversary should not be able to modify a packet header to change the packet's path without detection. The adversary should not learn information about forwarding nodes' location or the total number of forwarding nodes on a path.
- Payload secrecy and end-to-end integrity. Without compromising end hosts, an adversary cannot learn any information from the data payload except for its length and timing among sequences of packets. The adversary cannot alter any of the packet contents either.

1.3.2 Desired Performance Properties

To scale to the Internet and support diverse applications like teleconferencing, bulk downloading, and video streaming, the proposed systems should also satisfy the following three performance properties:

- Low path stretch factor. The path stretch factor is the increase in the number of AS hops normalized by the original path length used in non-anonymous protocols (e.g., AS-level paths in BGP). It is desirable to reduce the path stretch factor because an increased number of AS hops results in added propagation latency.
- High forwarding throughput. Routers should offer throughput comparable to the forwarding rates of current commodity routers. Specifically, the cryptographic operations required should not become a performance bottleneck when routers process data packets.

- Scalability. To avoid the state exhaustion problem, the proposed system should allow routers to minimize the amount of per-flow state maintained, or even keep no per-flow state, increasing scalability and reducing the state-based attack surface.

1.3.3 Categorization of Threat Model

Adversary's goal. The goal of an adversary is to conduct mass surveillance, i.e., de-anonymize end hosts of communication on a large scale. Specifically, the adversary aims to reduce the anonymity set size of senders or receivers of anonymous communication observed across controlled Autonomous Systems (AS).

Adversary's capability. The thesis considers three different levels of adversaries' capability as follows.

1. Local adversary. A local adversary is able to compromise a single AS on the path between source and destination hosts. For sender anonymity, the adversary can also compromise the destination host. For receiver anonymity, the adversary can compromise the source host. By compromising a host or an AS, the adversary learns all cryptographic keys and settings, observes all traffic that traverses the compromised entity, and is able to control how the entity behaves including redirecting traffic, fabricating, replaying, and modifying packets.

2. Global adversary with limited computation/storage power. A global adversary can compromise multiple ASes. The attacker could match packets by the bit patterns efficiently on a large scale. However, due to the limited computation/storage power, large-scale traffic analysis attacks [46, 53, 84,96,101,102,103,104,112,118,127,133,133,187,189,189,199,200] are beyond the capability of such an adversary. Current popular onion routing protocols, such as Tor [79], consider this type of adversary.

3. Global adversary with computation/storage power sufficient for traffic analysis. In addition to the previous adversary's capability, an adversary of this type can collect enough computing/storage power to conduct passive and active traffic analysis attacks on a large scale.

1.4 Thesis Outline

We summarize the main topic of each chapter as follows.

- Chapter 1 is the introduction. We discuss that anonymity, no matter how controversial, is vital to the society, especially in the Internet age. We describe current threats to anonymity, existing anonymity services, and why these services are unsatisfactory. We also introduce the concept of an infrastructure-based anonymous communication system, as well as its benefits and challenges.

- Chapter 2 reviews existing overlay-based and infrastructure-based anonymity systems, and different kinds of traffic analysis attacks.

- Chapter 3 presents HORNET, a high-speed onion routing protocol at the network layer [57]. HORNET is directly inspired by today's most popular anonymity system, Tor. However, as an infrastructure-based anonymity protocol, the primary challenge of HORNET is how to construct an onion routing protocol without using per-flow state. HORNET features a new onion routing protocol design with packet-carried state to eliminate per-flow state. Additionally, by implementing HORNET by using Data Plane Developer Kit (DPDK) and hardware-accelerated cryptographic instructions, we demonstrate that a HORNET router is able to offer high performance packet processing.

- Chapter 4 presents PHI, a path-hidden lightweight anonymity protocol at the network layer [60]. PHI belongs to a particular category of the lightweight infrastructure-based anonymity protocol. A lightweight anonymity protocol sacrifices anonymity guarantees for higher performance. We identify two attacks that shrink the anonymity set sizes of the existing lightweight anonymity protocols, i.e., LAP and Dovetail. PHI proposes a path-hidden packet header format and a back-off method to establish an end-to-end path without requiring control the path traversed.

- Chapter 5 presents TARANET, a traffic-analysis resistant anonymity protocol at the network layer [58]. TARANET aims to answer the question whether it is possible to simultaneously resist traffic-analysis attacks and achieve high performance. The fundamental guarantee behind TARANET is to maintain a constant transmission rate on all traversed links of a path. To accommodate packet loss and network jitter that can deplete the packet queue on an intermediate router and break the constant-rate transmission, TARANET proposes a new method that allows a packet to be split into two packets to refill the packet queues on traversed routers.

- Chapter 6 discusses the out-of-scope attacks and incremental deployment strategies for the three proposed protocols.

- Chapter 7 reviews the related work about anonymity systems and methods of applying chaff traffic.

- Chapter 8 concludes the dissertation. We also cover future directions to improve the proposed infrastructure-based anonymous communication systems.

Chapter 2

Background

In this chapter, we first present a concise discussion about the current overlay-based application-layer anonymity systems. We then introduce the concept of network-layer anonymity systems, and we discuss the proposed lightweight anonymity systems at the network layer. Finally, we summarize different categories of the traffic analysis attacks that this thesis considers.

2.1 Overlay-based Application-layer Anonymity Systems

The study of anonymous communication began with Chaum's proposal for Mix networks [55]. Since then, message-based mix systems have been proposed and deployed [70, 71, 99, 131]. These systems can withstand an active adversary and a significant fraction of compromised relays, but rely on expensive asymmetric primitives, message batching, and mixing. Thus, they suffer from large computational overhead and high latency and are suitable for supporting delay-tolerant applications like emails.

Later, low-latency onion routing systems [48, 49, 79, 153] were proposed to support interactive traffic efficiently. The low-latency onion routing systems only require symmetric cryptographic operations during packet forwarding and refrain from using batching and mixing that incurs delays. Nevertheless, the low-latency onion routing systems are vulnerable to end-to-end confirmation attacks [118], thus offering weaker anonymity guarantees compared to mix networks.

To achieve strong anonymity and good performance, the research community proposed protocols [88, 115, 116, 163] that offer traffic analysis resistance and higher performance than original mix networks. In this section, we categorize the relevant anonymity systems into low-latency onion routing systems and traffic-analysis resistant anonymity systems.

2.1.1 Low-latency Onion Routing Systems

An onion routing system is composed of volunteered servers that are willing to forward traffic for others. The general idea is that a user can pick a sequence of servers to route his/her traffic for anonymity. During packet forwarding, each server will independently apply encryption/decryption to traversing packets, thus preventing an adversarial observer from linking an incoming packet to an outgoing one.

Low-latency onion routing systems [11, 13, 48, 79] concentrate on reducing forwarding delay and enabling interactive applications. In general, low-latency onion routing systems group messages into sessions. At the start of each session, a user sets up the session through chosen servers by negotiating a symmetric key with each server. When forwarding a data packet, the server uses only symmetric cryptographic operations to encrypt/decrypt the packet. Due to lack of mechanisms to conceal traffic patterns across packets within a session, low-latency onion routing systems are typically vulnerable to end-to-end confirmation attacks discussed in Section 2.3. Because low-latency onion routing systems are widely used, people often refer to a low-latency onion routing system as an "onion routing system".

For example, the Tor network [79], the most popular low-latency onion routing system, consists of voluntarily-contributed servers named onion routers. To anonymity his/her traffic, a Tor user picks three onion routers and establishes a session called circuit. During the negotiation, each onion router obtains a symmetric key shared with the sender. All packets belong to the circuit are decrypted (for the direction from the sender to the receiver) or encrypted (for the direction from the receiver to the sender) during packet forwarding using the per-hop symmetric key. Like other low-latency onion routing systems, Tor is demonstrated to be vulnerable to confirmation attacks [41, 134, 138].

2.1.2 Traffic-analysis Resistant Anonymity Systems

There are two categories of anonymity systems that resist traffic analysis: Mix networks and Dining Cryptographer (DC) networks. David Chaum pioneered anonymous communication systems by proposing to route messages through a sequence of mix servers and to use layered encryption/decryption to prevent linking incoming and outgoing messages by bit patterns [56]. To defeat traffic analysis, a mix server also groups incoming messages into batches and mixes the messages within each batch by randomizing the order. Because batching introduces substantial delays and mixing alters message orders, the Mix network is suitable for delay-tolerant applications, such as anonymous remailers [70, 98, 131].

To reduce the delay caused by batching messages, the research community proposes to add chaff traffic (sometimes called cover traffic or dummy traffic) to conceal traffic's meta-data. ISDN-Mix first proposes to insert constant-rate chaff traffic into time-sliced channels and broadcast incoming calls on subscriber links

on an ISDN network [109, 145]. Brian et al. additionally suggest probabilistically dropping chaff traffic to provide stronger defense against timing analysis [118]. $\mathcal{P}^5$ [163] and Tarzan [88] are P2P networks that thwart traffic analysis by transmitting payload and chaff messages between peers at total fixed rates. Loopix [147] proposes to inject chaff traffic following the Poisson process to decrease the probability of linking input and output messages.

However, adding chaff traffic incurs additional bandwidth overhead. In fact, balancing the tradeoff between message delays and chaff traffic overhead is still an open research challenge. Both Java Anonymous Proxy (JAP) [14] and the Freedom Network [48] forfeit the traffic-analysis resistance property by abandoning chaff traffic due to the additional overhead.

There are also traffic-analysis resistant systems that use chaff traffic and are designed to serve specific applications. Aqua advocates a heterogeneous architecture to defeat traffic analysis attacks for peer-to-peer downloading [116]. End hosts form anonymity sets by sending their messages at a fixed rate in a synchronized manner at the network edge, while the network core uses chaff traffic to maintain constant-rate transmission and multi-path connections to avoid hotspot for file sharing. Herd [115] also uses a hybrid architecture similar to Aqua for VoIP services. Compared to Aqua, Herd uses network coding to improve scalability at the network edge and has shorter path length within the network core because of the constant-rate transmission nature of VoIP calls. Drac [69] forms circuits using social network links within a peer-to-peer network to anonymize VoIP traffic. Each node in Drac adds chaff traffic according to past observed packet schedules. Drac also uses full hiding strategies to hide connection and disconnection events in VoIP calls.

The DC network [54, 95] leverages broadcast channels and cryptographic primitives to guarantee anonymity with a very strong adversary model, where an adversary can compromise all nodes but one. However, the cryptographic operations required by the DC network are computationally expensive. Forming broadcast channels among participants limits the scalability of the DC network. Dissent [193] improves the DC network by providing message integrity and one-on-one correspondence between senders and messages to detect Denial-of-Service (DoS) attacks. However, Dissent still suffers from the same computation and scalability problems of the DC network. Riposte [62] improves the scalability by using the distributed point function, a technique developed in the context of Private Information Retrieval (PIR), to reduce the per-message bandwidth overhead of each client within a broadcast channel.

2.2 Network-layer Anonymity Protocols

Recent research [57, 105, 156] proposes *network-layer anonymity systems* that incorporate anonymous communication as a service of network infrastructures in the Internet and next-generation network architectures [92, 195, 198]. The underlying assumption of a network-layer anonymity system is that routers in Autonomous Systems (AS) can conduct efficient cryptographic operations when forwarding packets to provide anonymity for end hosts. This processing would happen on (software) routers. Additionally, a network-layer anonymity system uses direct forwarding paths rather than reroute packets through overlay networks as in Tor [79].

Besides anonymity, the basic design goals for a network-layer anonymity system are scalability and performance. Regarding scalability, a network-layer anonymity system minimizes the amount of state kept on network routers who possess limited high-speed memory. Regarding performance, a network-layer anonymity system should offer low latency and high throughput.

2.2.1 Lightweight Anonymity Systems

The first network-layer anonymous communication system is the so-called *lightweight* anonymity system, which aims to defeat a single malicious service provider by hiding forwarding information within packet headers. Compared to onion routing schemes like Tor, a lightweight anonymity system avoids conduct cryptographic operations over the payload, trading security with efficiency.

Hsiao et al.'s pioneering work, LAP [105], first covers the question how to design an efficient and scalable network-layer anonymity system whose packet headers hide network location information. To achieve scalability, a LAP forwarding node maintains no per-flow forwarding state. Instead, each LAP packet carries forwarding state. Each node maintains a local secret key and uses the key to decrypt the packet-carried state to retrieve the forwarding state. Because LAP reveals the destination addresses when setting up packet headers, LAP relies on the strong assumption that the first-hop node is benign, which is not always the case, especially when users are traveling or using public Internet access points.

Dovetail [156] removes the assumption of trusting the first-hop node by introducing a helper node during the packet header setup. The sender encrypts the destination address using the helper node's public key and sets up a half path to the helper node. The helper node decrypts the destination address and creates the second half path to the destination. To reduce the path stretch factor, the sender ensures the first half path and the second half path intersect at a node called the dovetail node. The sender combines the two half paths by removing nodes between the dovetail node and the helper node.

However, both LAP and Dovetail packet header formats leak information about the total number of

hops between the sender and the receiver and the number of hops between the sender/receiver and a forwarding node. Combined with knowledge about the network topology, an on-path adversarial node can use the leaked information to reduce the sender's (or receiver's) anonymity set.

	Protocol	Throughput	No Trust in First Hop Node	Hiding	Topology Information	No Need for Controlled Path	Bandwidth Efficiency	Low Latency	No Sender Coordination	Scalability	Bit-Pattern Unlinkability	Traffic Analysis Resistance
Network	Lap [105]	≈100 Gbps†	○	○	●	●	●	◑	●	○	○	
	Dovetail [156]	≈100 Gbps†	●	○	○	●	●	◑	●	○	○	
	PHI	≈100 Gbps†	●	●	●	●	●	◑	●	○	○	
	HORNET	92 Gbps†	●	●	●	●	●	◑	●	●	○	
	TARANET	50 Gbps†	●	●	●	◑	●	◑	●	●	◑	
Overlay	Tor [79]	1 Gbps†	●	●	●	●	◕	◑	◑	●	○	
	Mixminion [70]	6.4 Mbps	●	●	●	●	○	◑	◑	●	●	
	Dissent [193]	160 Kbps	●	●	●	◔	◑	◑	○	●	●	
	$\mathcal{P}^5$ [163]	1 Mbps	●	●	●	◑	◑	●	◑	●	◑	
	Tarzan [88]	60 Mbps	●	●	●	◑	◑	●	◑	●	◑	
	Aqua [116]	7.68 Mbps	○	●	●	◑	◑	○	◑	●	◑	
	Herd [115]	192 Kbps	○	●	●	◔	◑	○	◑	●	◑	

Table 2.1: Comparison of anonymous communication protocols. †As per our evaluation. Other results taken from original publication.

2.3 Traffic Analysis Attacks

Traffic analysis aims to identify communicating endpoints based on *metadata* such as volume, traffic patterns, and timing. The literature broadly classifies traffic analysis techniques into passive and active, depending on whether the adversary manipulates traffic.

2.3.1 Passive Attacks

Flow dynamics matching An adversary eavesdropping on traffic at two observation points (including an adversary observing the ingress and egress traffic of a single node) can try to detect whether (some of) the packets seen at the observation points belong to the same flow by searching for similarities among the dynamics of all observed flows [118, 127, 133, 200]. For example, the adversary can monitor packet inter-arrival times, flow volume [46], or on/off flow patterns [189, 199].

Template attacks An adversary can construct a database of traffic patterns (*templates*) obtained by accessing known websites or other web-services through the anonymous communication system. When

eavesdropping on the traffic of a client, the adversary compares the observed flows with the patterns stored in the database, and if a match is found the adversary is able to guess the website or web-service accessed by the client with high probability [96, 112, 187].

Network statistics correlation Another possible attack consists in monitoring network characteristics of different parts of the network and comparing them to the characteristics of targeted anonymized flows. For instance, by comparing the round-trip time (RTT) of a target bidirectional flow with the RTTs measured to a large set of network locations, an adversary can identify the probable network location of an end host in case the RTT of the flow showed strong correlation with the RTT to one of the monitored network locations [101]. Similarly, by recording the throughput (over time) of a unidirectional flow and comparing it with the throughput to various network locations, the adversary can guess the end host's location [127].

2.3.2 Active Attacks

Active traffic analysis uses similar techniques as passive traffic analysis, but it additionally involves traffic manipulation by the adversary, in particular, packet delaying and dropping, to introduce specific patterns. Chakravarty et al. [52] show that active analysis can have high success rates even when working with aggregate Netflow data instead of raw packet traces.

Flow dynamics modification By modifying the flow dynamics (inter-packet timings), the adversary can add a single bit *watermark* (or *tag*) to the flow, which the adversary is then able to detect when observing the flow at another point in the network [102, 104, 189]. This attack is known as *flow watermarking*. A similar attack, called *flow fingerprinting*, enables an adversary to encode multiple-bit information into the flow dynamics, which can later be extracted from the same flow seen at another point in the network [103]. For both attacks, depending on the coding technique, flows may require more or fewer packets for the watermark/fingerprint to be reliably identified within the network.

Clogging Attacks Flow dynamics modification requires that the adversary control multiple observation points in the network. Clogging attacks are similar, but the adversary only needs to be able to observe the target flow at a single network location. For these attacks, the adversary causes network congestion [84, 133], or fluctuation [53] at other nodes in the network, and then observes whether these actions affect the observed target flow. If so, it is likely that the target flow traverses the nodes at which congestion/fluctuation happens.

Chapter 3

HORNET: High-speed Onion Routing Protocol at the Network Layer

In this chapter, we present HORNET (High-speed Onion Routing at the NETwork layer), a highly-scalable anonymity system that leverages the next-generation Internet architecture design. HORNET offers payload protection by default, and can defend against attacks that exploit multiple network observation points. HORNET is designed to be highly efficient: it can use short paths offered by underlying network architectures, rather than the long paths due to global redirection; additionally, instead of keeping state at each relay, connection state (including, e.g., onion layer decryption keys) is carried within packet headers, allowing intermediate nodes to quickly forward traffic without per-packet state lookup.

While this chapter proposes and evaluates a concrete anonymity system, a secondary goal herein is to broadly re-think the design of low-latency anonymity systems by envisioning networks where anonymous communication is offered as an in-network service to all users. For example, what performance trade-offs exist between keeping anonymous connection state at relays and carrying state in packets? If routers perform anonymity-specific tasks, how can we ensure that these operations do not impact the processing of regular network traffic, especially in adversarial circumstances? And if the network architecture should provide some support for anonymous communication, what should that support be? Throughout this chapter we consider these issues in the design of our own system, and provide intuition for the requirements of alternative network-level anonymity systems.

Specifically, our contributions are the following:

- We design and implement HORNET, an anonymity system that uses source-selected paths and shared keys between endpoints and routers to support onion routing. Unlike other onion routing implementations, HORNET routers do not keep per-flow state or perform computationally expen-

sive operations for data forwarding, allowing the system to scale.

- We analyze the security of HORNET, showing that it can defend against passive attacks, and certain types of active attacks. HORNET provides stronger security guarantees than existing network-level anonymity systems.
- We evaluate the performance of HORNET, showing that its anonymous data processing speed is close to that of LAP and Dovetail (up to 93.5 Gb/s on a 120 Gb/s software router). This performance is comparable with that of today's high-end commodity routers [6].

The material presented here is closely related to the paper published at CCS'15 [57].

3.1 Problem Definition

We aim to design a network-level anonymity system to frustrate adversaries with mass surveillance capabilities. Specifically, an adversary observing traffic traversing the network should be unable to link (at large scale) pairs of communicating hosts . This property is known as relationship anonymity [144].

We define *sender anonymity* as a communication scenario where anonymity is guaranteed for the source, but the destination's location is public (e.g., web sites for The Guardian or Der Spiegel). We define *sender-receiver anonymity* as a scenario where the anonymity guarantee is extended to the destination (e.g., a hidden service that wishes to conceal its location). Sender-receiver anonymity therefore offers protection for both ends, implying sender anonymity. Depending on users' needs, HORNET can support either sender anonymity or sender-receiver anonymity.

Since our scheme operates at the network layer, network location is the only identity feature we aim to conceal. Exposure of network location or user identity at upper layers (e.g., through TCP sessions, login credentials, or browser cookies) is out of scope for this work.

3.1.1 Network Model

We consider that provisioning anonymous communication between end users is a principal task of the network infrastructure. The network's anonymity-related infrastructures, primarily routers, assist end users in establishing temporary *anonymous sessions* for anonymous data transmission.

We assume that the network layer is operated by a set of nodes. Each node cooperates with sources to establish anonymous sessions to the intended destinations, and processes anonymous traffic within the created sessions. We require that the routing state of a node allows it to determine only the next hop. In particular, the destination is only revealed to the last node and no others. This property can be satisfied

by IP Segment Routing [19], Future Internet Architectures (FIAs) like NIRA [195] and SCION [142,198], or Pathlets [92]. In practice, our abstract notion of a node could correspond to different entities depending on the architecture on which HORNET is built. For instance, in NIRA and SCION, a node corresponds to an Autonomous System (AS); in Pathlets, a node maps to a *vnode*.

Path and certificate retrieval A path is the combination of routing state of all nodes between the source and the intended destination. We assume the underlying network architecture provides a mechanism for a source to obtain such a path to a given destination. Additionally, we assume that the same mechanism allows the source to fetch the public keys and certificates[1] of on-path nodes. Note that the mechanism should be privacy-preserving: the source should not reveal its network location or intent to communicate with a destination by retrieving paths, public keys, and certificates. In Chapter 6.2, we further discuss how to obtain required information anonymously in selected FIAs. While a general solution represents an important avenue for future work, it remains outside of our present scope.

Public key verification We assume that end hosts and on-path nodes have public keys accessible and verifiable by all entities. End hosts can retrieve the public keys of other end hosts through an out-of-band channel (e.g., websites) and verify them following a scheme like HIP [132], in which the end hosts can publish hashes of their public keys as their service names. Public keys of on-path nodes are managed through a public-key infrastructure (PKI). For example, the source node can leverage Resource Public Key Infrastructure (RPKI) [50] to verify the public keys of on-path nodes.

3.1.2 Threat Model

We consider an adversary attempting to conduct mass surveillance. Specifically, the adversary collects and maintains a list of "selectors" (e.g., targets' network locations, or higher-level protocol identifiers), which help the adversary trawl intercepted traffic and extract parts of it for more extensive targeted analysis [15]. An anonymity system should prevent an adversary from leveraging bulk communication access to select traffic that belongs to the targets. Thus an adversary has to collect and analyze all traffic and cannot reliably select traffic specific to targets unless it has access to the physical links adjacent to the targets.

We consider an adversary that is able to compromise a fraction of nodes on the path between a source and a destination. For sender anonymity, the adversary can also compromise the destination. For sender-receiver anonymity, the adversary can compromise at most one of the two end hosts. By compromising a node, the adversary learns all keys and settings, observes all traffic that traverses the compromised

[1] Depending on the underlying PKI scheme, the source might need to fetch a chain of certificates leading to a trust anchor to verify each node's public key.

node, and is able to control how the nodes behave including redirecting traffic, fabricating, replaying, and modifying packets.

However, we do not aim to prevent targeted de-anonymization attacks where an adversary invests a significant amount of resources on a single or a small set of victims. Like other low-latency schemes, we cannot solve targeted confirmation attacks based on the analysis of flow dynamics [118, 134, 160]. Defending against such attacks using dynamic link padding [188] would be no more difficult than in onion routing, although equally expensive. We defer the discussion and analysis of such measures to future work.

3.1.3 Desired Properties

HORNET is designed to achieve the following anonymity and security properties:

1. **Path information integrity and secrecy**. An adversary cannot modify a packet header to alter a network path without detection. The adversary should not learn forwarding information of uncompromised nodes, node's positions, or the total number of hops on a path.

2. **No packet correlation.** An adversary who can eavesdrop on multiple links in the network cannot correlate packets on those links by observing the bit patterns in the headers or payloads. This should hold regardless of whether the observed traffic corresponds to the same packet (at different points on the network), or corresponds to different packets from a single session.

3. **No session linkage.** An adversary cannot link packets from different sessions, even between the same source and destination.

4. **Payload secrecy and end-to-end integrity**. Without compromising end hosts, an adversary cannot learn any information from the data payload except for its length and timing among sequences of packets.

3.2 HORNET Overview

The basic design objectives for HORNET are *scalability* and *efficiency*. To enable Internet-scale anonymous communication, HORNET intermediate nodes must avoid keeping per-session state (e.g., cryptographic keys and routing information). Instead, session state is offloaded to end hosts, who then embed this state into packets such that each intermediate node can extract its own state as part of the packet forwarding process.

Offloading the per-session state presents two challenges. First, nodes need to prevent their offloaded state from leaking information (e.g., the session's cryptographic keys). To address this, each HORNET node maintains a local secret to encrypt the offloaded per-session state. We call this encrypted state a *Forwarding Segment* (FS). The FS allows its creating node to dynamically retrieve the embedded information (i.e., next hop, shared key, session expiration time), while hiding this information from unauthorized third parties.

The second challenge in offloading the per-session state is to combine this state (i.e., the FSes) in a packet in such a way that each node is able to retrieve its own FS, but no information is leaked about the network location of the end hosts, the path length, or a specific node's position on the path. Learning any of this information could assist in de-anonymization attacks (see Section 3.4.5). To address this challenge, the source constructs an *anonymous header* (AHDR) by combining multiple FSes, and prepends this header to each packet in the session. An AHDR grants each node on the path access to the FS it created, without divulging any information about the path except for a node's previous and next nodes (see Section 3.3.4.1).

For efficient packet processing, each HORNET node performs one Diffie-Hellman (DH) key exchange operation once per session during setup. For all data packets within the session, HORNET nodes use only symmetric cryptography to retrieve their state, process the AHDR and onion-decrypt (or encrypt) the payload. To reduce setup delay, HORNET uses only two setup packets within a single round trip between the source and the destination. Therefore, session setup only incurs $O(n)$ propagation delay in comparison to $O(n^2)$ by the telescopic setup method used in Tor (where n is the number of anonymity nodes traversed on the path). While for Tor the default value of n is 3, for HORNET n might be as large as 14 (4.1 in the average case, and less or equal to 7 in over 99% of cases [12]), which emphasizes the need to optimize setup propagation delay.

3.2.1 Sender Anonymity

Anonymous sessions between a source and a destination require the source to establish state between itself and every node on the path. The state will be carried in subsequent data packets, enabling intermediate nodes to retrieve their corresponding state and forward the packet to the next hop. We now describe how the state is collected without compromising the sender's anonymity, and how this state is used to forward data packets.

Setup phase To establish an anonymous session between a source S and a public destination D, S uses a single round of Sphinx [71], a provably secure mix protocol (an overview of Sphinx is given in Section 3.3.3.1). This round consists of two Sphinx packets (one for the forward path and one for the

backward path) each of which will anonymously establish shared symmetric keys between S and every node on that path. For HORNET, we extend the Sphinx protocol to additionally anonymously collect the forwarding segments (FSes) for each node. Our modified Sphinx protocol protects the secrecy and integrity of these FSes, and does not reveal topology information to any node on the path. We note that using Sphinx also for data forwarding would result in low throughput due to prohibitively expensive per-hop asymmetric cryptographic operations. Therefore, we use Sphinx only for session setup packets, which are amortized over the subsequent data transmission packets. We explain the details of the setup phase in Section 3.3.3.

Data transmission phase Having collected the FSes, the source is now able to construct a forward AHDR and a backward AHDR for the forward and backward paths, respectively. AHDRs carry the FSes which contain all state necessary for nodes to process and forward packets to the next hop. When sending a data packet, the source onion-encrypts the data payload using the session's shared symmetric keys, and prepends the AHDR. Each node then retrieves its FS from the AHDR, onion-decrypts the packet and forwards it to the next hop, until it reaches the destination. The destination uses the backward AHDR (received in the first data packet[2]) to send data back to S, with the only difference being that the payload is encrypted (rather than decrypted) at each hop. We present the details of the data transmission phase in Section 3.3.4.

3.2.2 Sender-Receiver Anonymity

Sender-receiver anonymity, where neither S nor D knows the other's location (e.g., a hidden service), presents a new challenge: since S does not know D's location (and vice versa), S cannot retrieve a path to D, precluding the establishment of state between S and nodes on the path to D as described in Section 3.2.1.

A common approach to this problem (as adopted by Tor[3], LAP, and Dovetail) is to use a public *rendezvous point* (RP) to forward traffic between S and D without knowing either S or D. This solution would also work for HORNET, but would require RPs to maintain per-session state between sources and destinations. For instance, when receiving a packet from S, an RP needs the state to determine how to send the packet to D. Maintaining per-session state on RPs increases complexity, bounds the number of receivers, and introduces a state exhaustion denial-of-service attack vector.

[2]If the first packet is lost the source can simply resend the backward AHDR using a new data packet (see Section 3.3.4).

[3]Tor additionally uses an introduction point, which enables S to negotiate a rendezvous point with D. This design provides additional scalability and attack resistance [79], but increases the delay of setting up a session. HORNET's design favors simplicity and performance, but nothing fundamentally prevents HORNET from using Tor's approach.

Nested ahdrs. Our proposal for sender-receiver anonymity requires no state to be kept at the RP by nesting the necessary state for RPs to forward a packet within the packet's header: a forward AHDR from S to a RP will include the AHDR from the RP to D; a backward AHDR from D to a RP will include the AHDR from the RP back to S.

Briefly, to establish a HORNET session between S and D keeping both parties hidden from each other, D selects a public rendezvous point R and completes a HORNET session setup between D and R. D publishes $\text{AHDR}_{R \to D}$ to a public directory. Note that this AHDR leaks no information about D's location and can only be used to send data to D through R within a specific time window.

When S wants to send traffic to D, S retrieves (from a public directory) $\text{AHDR}_{R \to D}$. S then establishes a HORNET session between S and R and constructs a nested AHDR with $\text{AHDR}_{R \to D}$ inside $\text{AHDR}_{S \to R}$. Thus, when R receives a packet from S, R can retrieve $\text{AHDR}_{R \to D}$ from $\text{AHDR}_{S \to R}$ and forward the packet to D. S also includes $\text{AHDR}_{R \to S}$ in the data payload of the first data packet to D, allowing D to create a return path to S.

One of the advantages of our scheme is that any node on the network can serve as a rendezvous point. In fact, multiple points can be selected and advertised, allowing the source to pick the RP closest to it. Moreover, once a HORNET session has been established, S and D can negotiate a better (closer) RP (e.g., using private set intersection [89]). A disadvantage of the nested AHDR technique is that it doubles the size of the header.

For space reasons, the formal protocol details and evaluation sections focus on sender anonymity only. Details of sender-receiver anonymity can be found in the technical report [59].

3.2.3 Packet Structure

HORNET uses two types of packets: *setup packets* and *data packets* (see Figure 3.1). Both types of packets begin with a common header (CHDR) which describes the packet type, the length of the longest path that the session supports, and a type-specific field. For session setup packets, the type-specific field contains a value EXP which indicates the intended expiration time of the session. For data packets, the specific value is a random nonce generated by the sender used by intermediate nodes to process the data packet.

Session setup packets include a nested Sphinx packet and an FS payload. Data packets carry an AHDR and an onion-encrypted data payload. We explain each field in detail in Section 3.3.

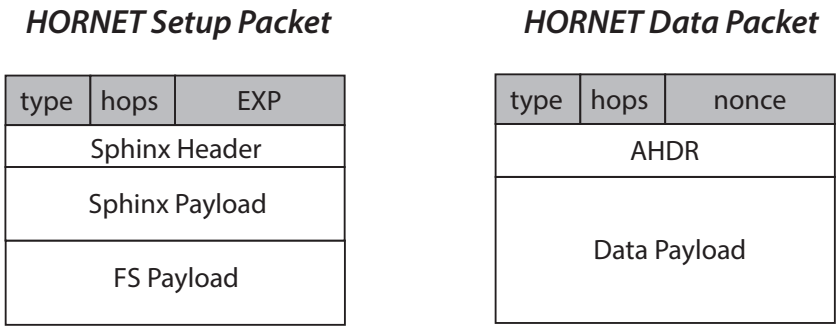


Figure 3.1: HORNET packet formats. For both setup packet and data packet, the shaded fields represent the common header (CHDR).

3.3 Formal Protocol Description

We now describe the details of our protocol, focusing on sender anonymity. We begin with notation (Section 3.3.1) and initialization requirements (Section 3.3.2). We then describe the establishment of anonymous communication sessions (Section 3.3.3) and data transmission (Section 3.3.4).

3.3.1 Notation

Let k be the security parameter used in the protocol. For evaluation purposes we consider $k = 128$. $\mathcal{G}$ is a prime order cyclic group of order q ($q \sim 2^{2k}$), which satisfies the Decisional Diffie-Hellman Assumption. $\mathcal{G}^*$ is the set of non-identity elements in $\mathcal{G}$ and g is a generator of $\mathcal{G}$. Throughout this section we use the multiplicative notation for $\mathcal{G}$.

Let r be the maximum length of a path, i.e., the maximum number of nodes on a path, including the destination. We denote the length of an FS as $|FS|$ and the size of an AHDR block, containing an FS and a MAC of size k, as $c = |FS| + k$.

HORNET uses the following cryptographic primitives:

- MAC : $\{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$: Message Authentication Code (MAC) function.
- PRG0, PRG1, PRG2 : $\{0,1\}^k \rightarrow \{0,1\}^{rc}$: Three cryptographic pseudo-random generators.
- PRP : $\{0,1\}^k \times \{0,1\}^a \rightarrow \{0,1\}^a$: A pseudo-random permutation, implementable as a block cipher. The value of a will be clear from the context.
- ENC : $\{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{mk} \rightarrow \{0,1\}^{mk}$: Encryption function, with the second parameter being the Initialization Vector (IV) (e.g., stream cipher in CBC mode). m is a positive integer denoting the number of encrypted blocks.

- $\mathsf{DEC} : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{mk} \to \{0,1\}^{mk}$: Decryption function, inverse of ENC.
- $h_{\mathsf{op}} : \mathcal{G}^* \to \{0,1\}^k$: a family of hash functions used to key op, with $\mathsf{op} \in \{\mathsf{MAC}, \mathsf{PRG0}, \mathsf{PRG1}, \mathsf{PRP}, \mathsf{ENC}, \mathsf{DEC}\}$.

We denote by $\textsc{rand}(a)$ a function that generates a new uniformly random string of length a.

Furthermore, we define the notation for bit strings. 0^a stands for a string of zeros of length a. $|\sigma|$ is the length of the bit string σ. $\sigma_{[a...b]}$ represents a substring of σ from bit a to bit b, with sub-index a starting from 0; $\sigma_{[a...end]}$ indicates the substring of σ from bit a till the end. ε is the empty string. $\sigma \parallel \sigma'$ is the concatenation of string σ and string σ'. We summarize protocol notation and typical values for specific parameters in Table 5.2.

In the following protocol description, we consider a source S communicating with a destination D using forward path p^f traversing nodes $n_0^f, n_1^f, \ldots, n_{l^f-1}^f$ and backward path p^b traversing nodes $n_0^b, n_1^b, \ldots, n_{l^b-1}^b$, with $l^f, l^b \leq r$, where n_0^f and $n_{l^b-1}^b$ are the nodes closest to the source. Without loss of generality, we let the last node on the forward path $n_{l^f-1}^f = D$ and refer to the destination by these two notations interchangeably. In general we use $dir \in \{f, b\}$ as superscripts to distinguish between notation referring to the forward and backward path, respectively. Finally, to avoid redundancy, we use $\{sym_i^{dir}\}$ to denote $\{sym_i^{dir} | 0 \leq i \leq l^{dir} - 1\}$, where sym can be any symbol.

3.3.2 Initialization

Suppose that a source S wishes to establish an anonymous session with a public destination D. First, S anonymously obtains (from the underlying network) paths in both directions: a forward path $p^f = \{R_0^f, R_1^f, \cdots, R_{l^f-1}^f\}$ from S to D and a backward path $p^b = \{R_0^b, R_1^b, \cdots, R_{l^b-1}^f\}$ from D to S. R_i^{dir} denotes the routing information needed by the node n_i^{dir} to forward a packet. S also anonymously retrieves and verifies a set of public keys $g^{x_{n_i^{dir}}}$ for the node n_i^{dir} on path p^{dir} (see Section 3.1.1). Note that g^{x_D} is also included in the above set (as $n_{l^f-1}^f = D$). Finally, S generates a random DH public/private key pair for the session: x_S and g^{x_S}. The per-session public key g^{x_S} is used by the source to create shared symmetric keys with nodes on the paths later in the setup phase. S locally stores $\left\{(x_S, g^{x_S}), \left\{g^{x_{n_i^{dir}}}\right\}, p^{dir}\right\}$, and uses these values for the setup phase.

3.3.3 Setup Phase

As discussed in Section 3.2, in the setup phase, HORNET uses two Sphinx packets, which we denote by P❶ and P❷, to traverse all nodes on both forward and backward paths and establish per-session state with every intermediate node, without revealing S's network location. For S to collect the generated per-session state from each node, both Sphinx packets contain an empty FS payload into which each

Term	Definition
k	Security parameter (length of keys and MACs). $k = 128$ bits (16 B).
$\|FS\|$	Length of a forwarding segment (FS). $\|FS\| = 256$ bits (32 B).
c	Length of a typical block made of an FS and a MAC. $c = \|FS\| + k = 384$ bits (48 B) .
r	Maximum path length, including the destination. From our evaluation, $r = 7$.
S, D	Source and destination.
p^f, p^b	The forward path (from S to D) and the backward path (from D to S).
l^f, l^b	Lengths of the forward and backward path (l, when it is clear from the context to which path it refers). From our evaluation, $1 \leq l \leq 7$.
n_i^f, n_j^b	The i-th node on the forward path and the j-th node on the backward path, with $0 \leq i < l^f$ and $0 \leq j < l^b$.
g^{x_n}, x_n	Public/private key pair of node n.
s_i^f	Secret key established between S and node n_i^f.
R	Routing information, which allows a node to forward a packet to the next hop.
CHDR	Common header. First three fields of both setup packets and data packets (see Figure 3.1).
SHDR, SP	Sphinx header and payload.
P	FS payload, used to collect the FSes during the setup phase.
AHDR	Anonymous header, used for every data packet. It allows each node on the path to retrieve its FS.
O	Onion payload, containing the data payload of data packets.
EXP	Expiration time, included in each FS.

Table 3.1: HORNET protocol notation and typical values (where applicable).

intermediate node can insert its FS, but is not able to learn anything about, or modify, previously inserted FSes.

3.3.3.1 Sphinx Overview

Sphinx [71] is a provably-secure mix protocol. Each Sphinx packet allows a source node to establish a set of symmetric keys, one for each node on the path through which packets are routed. These keys enable each node to check the header's integrity, onion-decrypt the data payload, and retrieve the information to route the packet. Processing Sphinx packets involves expensive asymmetric cryptographic operations, thus Sphinx alone is not suitable to support high-speed anonymous communication.

Sphinx packets A Sphinx packet is composed of a Sphinx header SHDR and a Sphinx payload SP. The SHDR contains a group element ${y_i}^{dir}$ that is re-randomized at each hop. Each ${y_i}^{dir}$ is used as S's ephemeral public key in a DH key exchange with node n_i^{dir}. From this DH exchange, node n_i^{dir} derives a shared symmetric key s_i^{dir}, which it uses to process the rest of the SHDR and mutate ${y_i}^{dir}$. The rest of the SHDR is an onion-encrypted data structure, with each layer containing routing information and a MAC. The routing information indicates to which node the packet should be forwarded to next, and the MAC allows to check the header's integrity at the current node. The Sphinx payload SP allows end hosts to send confidential content to each other. Each intermediate node processes SP by using a pseudo-random permutation.

Sphinx core functions We abstract the Sphinx protocol into the following six functions:

- GEN_SPHX_HDR. The source uses this function to generate two Sphinx headers, SHDR^f and SHDR^b, for the forward and backward path, respectively. It also outputs the symmetric keys $\{s_i^{dir}\}$, each established with the corresponding node's public key ${g^x}_{n_i^{dir}}$.
- GEN_SPHX_PL_SEND. The function allows the source to generate an onion-encrypted payload SP^f encapsulating confidential data to send to the destination.
- UNWRAP_SPHX_PL_SEND. The function removes the last encryption layer added by GEN_SPHX_PL_SEND, and allows the destination to decrypt the SP^f.
- GEN_SPHX_PL_RECV. The function enables the destination to cryptographically wrap a data payload into SP^b before sending it to the source.
- UNWRAP_SPHX_PL_RECV. The function allows the source to recover the plaintext of the payload that the destination sent.

- PROC_SPHX_PKT. Intermediate nodes use this function to process a Sphinx packet, and establish symmetric keys shared with the source. The function takes as inputs the packet (SHDR, SP), and the node's DH public key $g^{x_{n_i^{dir}}}$. The function outputs the processed Sphinx packet $(\text{SHDR}', \text{SP}')$ and the established symmetric key s_i^{dir}.

3.3.3.2 Forwarding Segment

We extend Sphinx to allow each node to create a Forwarding Segment (FS) and add it to a data structure we name FS payload (see below). An FS contains a node's per-session state, which consists of a secret key s shared with the source, a routing segment R, and the session's expiration time EXP. To protect these contents, the FS is encrypted with a PRP keyed by a secret value SV known only by the node that creates the FS. A node seals and unseals its state using two opposite functions: FS_CREATE and FS_OPEN. They are defined as follows:

$$FS = \text{FS\_CREATE}(SV, s, R, \text{EXP}) = \mathsf{PRP}(h_{\mathsf{PRP}}(SV); \{s \parallel R \parallel \text{EXP}\}) \tag{3.1}$$

$$\{s \parallel R \parallel \text{EXP}\} = \text{FS\_OPEN}(SV, FS) = \mathsf{PRP}^{-1}(h_{\mathsf{PRP}}(SV); FS) \tag{3.2}$$

3.3.3.3 FS Payload

At the end of each HORNET setup packet is a data structure we call FS payload (see Figure 3.1). The FS payload is an onion-encrypted construction that allows intermediate nodes to add their FSes as onion-layers.

Processing the FS payload leaks no information about the path's length or about an intermediate node's position on the path. All FS payloads are padded to a fixed length, which is kept constant by dropping the right number of trailing bits of the FS payload before an FS is added to the front. Moreover, new FSes are always added to the beginning of the FS payload, eliminating the need for intermediate nodes to know their positions in order to process FS payloads.

An FS payload also provides both secrecy and integrity for the FSes it contains. Each node re-encrypts the FS payload after inserting a new FS and computes a MAC over the resulting structure. Only the source, with symmetric keys shared with each node on a path, can retrieve all the FSes from the FS payload and verify their integrity.

Functions There are three core functions for the FS payload: INIT_FS_PAYLOAD, ADD_FS, and RETRIEVE_FSES.

INIT_FS_PAYLOAD. A node initializes an FS payload by using a pseudo-random generator keyed with a symmetric key s to generate rc random bits:

$$P = \mathsf{PRG1}(h_{\mathsf{PRG1}}(s)) \tag{3.3}$$

where $c = |FS| + k$ is the size of a basic block of the FS payload (consisting of an FS and a MAC).

ADD_FS. Each intermediate node uses ADD_FS to insert its FS into the payload, as shown in Algorithm 1. First, the trailing c bits of the current FS payload, which are padding bits containing no information about previously added FSes, are dropped, and then the FS is prepended to the shortened FS payload. The result is encrypted using a stream cipher (Line 2) and MACed (Line 4). Note that no node-position information is required in ADD_FS, and verifying that the length of the FS payload remains unchanged is straightforward.

Algorithm 1 Add FS into FS payload.

1: **procedure** ADD_FS
   Input: s, FS, P_{in}
   Output: P_{out}
2: $P_{tmp} \leftarrow \left\{ FS \parallel P_{in[0..(r-1)c-1]} \right\} \oplus \mathsf{PRG0}(h_{\mathsf{PRG0}}(s))_{[k..end]}$
3: $\alpha \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s); P_{tmp})$
4: $P_{out} \leftarrow \alpha \parallel P_{tmp}$
5: **end procedure**

RETRIEVE_FSES. The source uses this function to recover all FSes $\{FS_i\}$ inserted into an FS payload P. RETRIEVE_FSES starts by recomputing the discarded trailing bits (Line 3) and obtaining a complete payload P_{full}. Thus, intuitively, this full payload is what would remain if no nodes dropped any bits before inserting a new FS. Afterwards, the source retrieves the FSes from P_{full} in the reverse order in which they were added by ADD_FS (see lines 6 and 8).

3.3.3.4 Setup Phase Protocol Description

Source processing With the input

$$I = \left\{ (x_S, g^{x_S}), \left\{ g^{x_{n_i^{dir}}} \right\}, p^{dir} \right\}$$

the source node S bootstraps a session setup in 5 steps:

1. S selects the intended expiration time EXP for the session and specifies it in the common header CHDR (see Section 3.2.3).[4]

[4] EXP must not become an identifier that allows matching packets of the same flow across multiple links. Since EXP does not

Algorithm 2 Retrieve FSes from FS payload.

procedure RETRIEVE_FSES
Input: P, s, $\{s_i\}$
Output: $\{FS_i\}$
  $P_{init} \leftarrow$ INIT_FS_PAYLOAD(s)
  $\psi \leftarrow P_{init[(r-l)c..rc-1]}$
    $\oplus\, \mathsf{PRG0}(h_{\mathsf{PRG0}}(s_0))_{[(r-l+1)c..end]} \parallel 0^c$
    $\oplus\, \mathsf{PRG0}(h_{\mathsf{PRG0}}(s_1))_{[(r-l+2)c..end]} \parallel 0^{2c}$
      $\cdots$
    $\oplus\, \mathsf{PRG0}(h_{\mathsf{PRG0}}(s_{l-2}))_{[(r-1)c..end]} \parallel 0^{(l-1)c}$
  $P_{full} = P \parallel \psi$
  for $i \leftarrow (l-1), \ldots, 0$ **do**
    check $P_{full_{[0..k-1]}} =$
      $\mathsf{MAC}(h_{\mathsf{MAC}}(s_i); P_{full_{[k..rc-1]}})$
    $P_{full} \leftarrow P_{full} \oplus (\mathsf{PRG0}(h_{\mathsf{PRG0}}(s_i)) \parallel 0^{(i+1)c})$
    $FS_i \leftarrow P_{full_{[k..c-1]}}$
    $P_{full} \leftarrow P_{full_{[c..end]}}$
  end for
end procedure

2. S generates the send and the reply Sphinx headers by:

$$\{\text{SHDR}^f, \text{SHDR}^b\} = \text{GEN\_SPHX\_HDR}(I, \text{CHDR}) \tag{3.4}$$

   The common header CHDR (see Figure 3.1) is passed to the function to extend the per-hop integrity protection of Sphinx over it. GEN_SPHX_HDR also produces the symmetric keys shared with each node on both paths $\{s_i^{dir}\}$.

3. In order to enable the destination D to reply, S places the reply Sphinx header SHDR^b into the Sphinx payload:

$$\text{SP}^f = \text{GEN\_SPHX\_PL\_SEND}(\{s_i^f\}, \text{SHDR}^b) \tag{3.5}$$

4. S creates an initial FS payload $P^f =$ INIT_FS_PAYLOAD(x_S).

5. S composes $\mathsf{P❶} = \{\text{CHDR} \parallel \text{SHDR}^f \parallel \text{SP}^f \parallel P^f\}$ and sends it to the first node on the forward path n_0^f.

Intermediate node processing An intermediate node n_i^f receiving a packet $\mathsf{P❶} = \{\text{CHDR} \parallel \text{SHDR}^f \parallel \text{SP}^f \parallel P^f\}$ processes it as follows:

change during setup packet forwarding, a coarser granularity (e.g., 10s) is desirable. In addition, the duration of the session should also have only a restricted set of possible values (e.g., 10s, 30s, 1min, 10min) to avoid matching packets within long sessions. For long-lived connections, the source can create a new session in the background before expiration of the previous one to avoid additional latency.

1. n_i^f first processes SHDR^f and SP^f in P❶ according to the Sphinx protocol (using PROC_SPHX_PKT). As a result n_i^f obtains the established symmetric key s_i^f shared with S, the processed header and payload $(\text{SHDR}^{f\prime}, \text{SP}^{f\prime})$ as well as the routing information R_i^f. During this processing the integrity of the CHDR is verified.

2. n_i^f obtains EXP from CHDR and checks that EXP is not expired. n_i^f also verifies that R_i^f is valid.

3. n_i^f generates its forwarding segment FS_i^f by using its local symmetric key SV_i^f to encrypt s_i^f, R_i^f, and EXP (see Equation 3.1):

$$FS_i^f = \text{FS\_CREATE}(SV_i^f, s_i^f, R_i^f, \text{EXP}) \tag{3.6}$$

4. n_i^f adds its FS_i^f into the FS payload P^f.

$$P^{f\prime} = \text{ADD\_FS}(s_i^f, FS_i^f, P^f) \tag{3.7}$$

5. Finally node n_i^f assembles the processed packet $\text{P❶} = \{\text{CHDR} \parallel \text{SHDR}^{f\prime} \parallel \text{SP}^{f\prime} \parallel P^{f\prime}\}$ and routes it to the next node according to the routing information R_i^f.

Destination processing As the last node on the forward path, D processes P❶ in the same way as the previous nodes. It first processes the Sphinx packet in P❶ and derives a symmetric key s_D shared with S, and then it encrypts per-session state, including s_D, into FS_D, and inserts FS_D into the FS payload.

After these operations, however, D moves on to create the second setup packet P❷ as follows:

1. D retrieves the Sphinx reply header using the symmetric key s_D:

$$\text{SHDR}^b = \text{UNWRAP\_SPHX\_PL\_SEND}(s_D, \text{SP}^f) \tag{3.8}$$

2. D places the FS payload P^f of P❶ into the Sphinx payload SP^b of P❷ (this will allow S to get the FSes $\{FS_i^f\}$):

$$\text{SP}^b = \text{GEN\_SPHX\_PL\_RECV}(s_D, P^f) \tag{3.9}$$

   Note that since D has no knowledge about the keys $\{s_i^f\}$ except for s_D, D learns nothing about the other FSes in the FS payload.

3. D creates a new FS payload $P^b = \text{INIT\_FS\_PAYLOAD}(s_D)$ to collect the FSes along the backward path.

4. D composes $\text{P❷} = \{\text{CHDR} \parallel \text{SHDR}^b \parallel \text{SP}^b \parallel P^b\}$ and sends it to the first node on the backward path, n_0^b.

The nodes on the backward path process P❷ in the exact same way nodes on the forward path processed P❶. Finally P❷ reaches the source S with FSes $\{FS_i^b\}$ added to the FS payload.

Post-setup processing Once S receives P❷ it extracts all FSes, i.e., $\{FS_i^f\}$ and $\{FS_i^b\}$, as follows:

1. S recovers the FS payload for the forward path P^f from SP^b:

$$P^f = \text{UNWRAP\_SPHX\_PL\_RECV}(\{s_i^b\}, \text{SP}^b) \tag{3.10}$$

2. S retrieves the FSes for the nodes on the forward path $\{FS_i^f\}$:

$$\{FS_i^f\} = \text{RETRIEVE\_FSES}(\{s_i^f\}, P^f) \tag{3.11}$$

3. S directly extracts from $P^b$the FSes for the nodes on the backward path $\{FS_i^b\}$:

$$\{FS_i^b\} = \text{RETRIEVE\_FSES}(\{s_i^b\}, P^b) \tag{3.12}$$

With the FSes for all nodes on both paths, $\{FS_i^f\}$ and $\{FS_i^b\}$, S is ready to start the data transmission phase.

3.3.4 Data Transmission Phase

Each HORNET data packet contains an anonymous header AHDR and an onion-encrypted payload O as shown in Figure 3.1. Figure 3.2 illustrates the details of an AHDR. The AHDR allows each intermediate node along the path to retrieve its per-session state in the form of an FS and process the onion-encrypted data payload. All processing of data packets in HORNET only involves symmetric-key cryptography, therefore supporting fast packet processing.

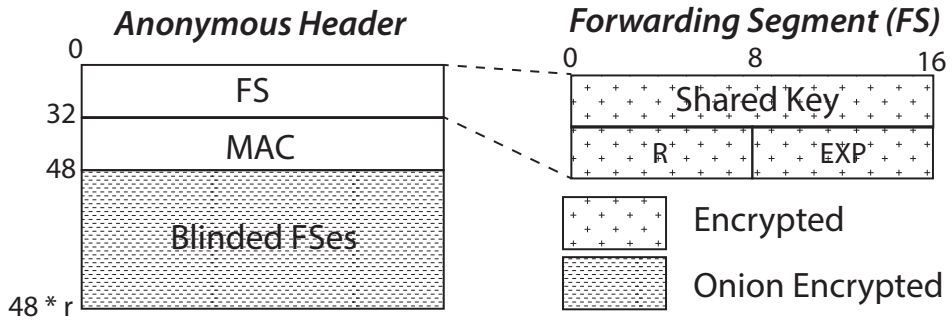


Figure 3.2: Format of a HORNET anonymous header with details of a forwarding segment (FS).

At the beginning of the data transmission phase, S creates two AHDRs, one for the forward path (AHDR^f) and one for the backward path (AHDR^b), by using FSes collected during the setup phase. AHDR^f enables S to send data payloads to D. To enable D to transmit data payloads back, S sends AHDR^b as payload in the first data packet. If this packet is lost, the source would notice from the fact that no reply is seen from the destination. If this happens the source simply resends the backward AHDR using a new data packet.

3.3.4.1 Anonymous Header

Like an FS payload, an AHDR is an onion-encrypted data structure that contains FSes. It also offers the same guarantees, i.e., secrecy and integrity, for the individual FSes it contains, for their number and for their order. Its functionalities, on the other hand, are the inverse: while the FS payload allows the source to collect the FSes added by intermediate nodes, the AHDR enables the source to re-distribute the FSes back to the nodes for each transmitted data packet.

Functions The life cycle of AHDRs consists of two functions: the header construction (CREATE_AHDR) and the header processing (PROC_AHDR). We begin with the description of PROC_AHDR since it is simpler, and its helps understand the construction of CREATE_AHDR.PROC_AHDR allows each intermediate node to verify the integrity of an incoming AHDR, and to check that the corresponding session has not expired. PROC_AHDR also retrieves the key s shared with the source, as well as the routing information R, from the FS of the node invoking the function. Finally, PROC_AHDR also returns the processed header AHDR$'$, which will be used by the next hop. The details of this function can be seen in Algorithm 3.

Algorithm 3 Process an AHDR.

1: **procedure** PROC_AHDR
   Input: SV, AHDR
   Output: s, R, AHDR$'$
2: $\{FS \parallel \gamma \parallel \beta\} \leftarrow$ AHDR
3: $\{s \parallel R \parallel \text{EXP}\} \leftarrow \text{FS\_OPEN}(SV, FS)$
4: **check** $\gamma = \mathsf{MAC}(h_{\mathsf{MAC}}(s); FS \parallel \beta)$
5: **check** $t_{curr} <$ EXP
6: AHDR$' \leftarrow \{\beta \parallel 0^c\} \oplus \mathsf{PRG2}(h_{\mathsf{PRG2}}(s))$
7: **end procedure**

Our AHDR construction resembles the Sphinx packet header construction [71]. For each path (forward and backward), CREATE_AHDR enables S to create an AHDR given the keys $\{s_i\}$ shared with each node on that path, and given the forwarding segments $\{FS_i\}$ of those nodes. All these keys and FSes are obtained during the setup phase (see Section 3.3.3). The details are shown in Algorithm 4. In essence, CREATE_AHDR is equivalent to a series of PROC_AHDR iterations performed in reverse. Initially, the paddings ϕ are computed, each of which is the leftmost part of an AHDR that results from the successive encryptions of the zero-paddings added in PROC_AHDR (ϕ_0 is the empty string since no padding has been added yet). Once the last padding is computed (the one for the AHDR received by the last hop, ϕ_{l-1}), the operations in PROC_AHDR are reversed, obtaining at each step the AHDRs as will be received by the nodes, from the last to the first. This also allows the computation of the per-hop MACs.

Algorithm 4 Anonymous header construction.

procedure CREATE_AHDR
Input: $\{s_i\}$, $\{FS_i\}$
Output: $(FS_0, \gamma_0, \beta_0)$
  $\phi_0 \leftarrow \varepsilon$
  for $i \leftarrow 0, \ldots, l-2$ **do**
    $\phi_{i+1} \leftarrow (\phi_i \parallel 0^c) \oplus \left\{\mathsf{PRG2}(h_{\mathsf{PRG2}}(s_i))_{[(r-1-i)c..end]}\right\}$
  end for
  $\beta_{l-1} \leftarrow \text{RAND}\left((r-l)c\right) \parallel \phi_{l-1}$
  $\gamma_{l-1} \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s_{l-1}); FS_{l-1} \parallel \beta_{l-1})$
  for $i \leftarrow (l-2), \ldots, 0$ **do**
    $\beta_i \leftarrow \left\{FS_{i+1} \parallel \gamma_{i+1} \parallel \beta_{i+1[0..(r-2)c-1]}\right\} \oplus \mathsf{PRG2}(h_{\mathsf{PRG2}}(s_i))_{[0..(r-1)c-1]}$
    $\gamma_i \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s_i); FS_i \parallel \beta_i)$
  end for
end procedure

3.3.4.2 Onion Payload

HORNET data payloads are protected by onion encryption. To send a data payload to the destination, the source adds a sequence of encryption layers on top of the data payload, one for each node on the forward path (including the destination). As the packet is forwarded, each node removes one layer of encryption, until the destination removes the last layer and obtains the original plaintext.

To send a data payload back to the source, the destination adds only one layer of encryption with its symmetric key shared with the source. As the packet is forwarded, each node on the backward path re-encrypts the payload until it reaches the source. With all the symmetric keys shared with nodes on the backward path, the source is capable of removing all encryption layers, thus obtaining the original data payload sent by the destination.

Functions Processing onion payloads requires the following two functions: ADD_LAYER and REMOVE_LAYER.

ADD_LAYER. The function's full form is:

$$\{O', IV'\} = \text{ADD\_LAYER}(s, IV, O) \tag{3.13}$$

Given a symmetric key s, an initial vector IV, and an input onion payload O, ADD_LAYER performs two tasks. First, ADD_LAYER encrypts O with s and IV:

$$O' = \mathsf{ENC}(h_{\mathsf{ENC}}(s); IV; O) \tag{3.14}$$

Then, to avoid making the IV an identifier across different links, ADD_LAYER mutates the IV for the next

node:

$$IV' = \mathsf{PRP}(h_{\mathsf{PRP}}(s); IV) \tag{3.15}$$

REMOVE_LAYER. The function is the inverse of ADD_LAYER, decrypting the onion payload at each step, and mutating the IV using the inverse permutation PRP^{-1} keyed with $h_{\mathsf{PRP}}(s)$. Its full form is the following:

$$\{O', IV'\} = \textsc{remove\_layer}(s, IV, O) \tag{3.16}$$

3.3.4.3 Initializing Data Transmission

To start the data transmission session, S generates $\textsc{ahdr}^f$ and $\textsc{ahdr}^b$ as follows:

$$\textsc{ahdr}^f = \textsc{create\_ahdr}(\{s_i^f\}, \{FS_i^f\}) \tag{3.17}$$

$$\textsc{ahdr}^b = \textsc{create\_ahdr}(\{s_i^b\}, \{FS_i^b\}) \tag{3.18}$$

S then sends $\textsc{ahdr}^b$ to D as payload of the first data packet (which uses $\textsc{ahdr}^f$), as specified in the following section.

3.3.4.4 Data Transmission Protocol Description

Source processing With $\textsc{ahdr}^f$, S can send a data payload P with the following steps:

1. S ensures that the session is not expired by checking that the current time $t_{curr} < \textsc{exp}$.
2. S creates an initial IV. With the shared keys $\{s_i^f\}$, S onion encrypts the data payload M by setting $O_{lf} = M$ and $IV_{lf} = IV$ and computing the following for $i \leftarrow (l^f - 1)..0$:

$$\{O_i, IV_i\} = \textsc{add\_layer}(s_D, IV_{i+1}, O_{i+1}) \tag{3.19}$$

3. S places IV_0 in the common header CHDR.
4. S sends out the resulting data packet $\{\textsc{chdr}, \textsc{ahdr}^f, O_0\}$.

Processing by intermediate nodes Each intermediate node n_i^f on the forward path processes a received data packet of the form $\{\textsc{chdr}, \textsc{ahdr}^f, O\}$ with its local secret key SV_i^f as follows:

1. n_i^f retrieves the key s_i^f shared with S and the routing information R_i^f from $\textsc{ahdr}^f$:

$$\{s_i^f, R_i^f, \textsc{ahdr}^{f\prime}\} = \textsc{proc\_ahdr}(SV_i^f, \textsc{ahdr}^f) \tag{3.20}$$

PROC_AHDR also verifies the integrity of AHDR, and checks that the session has not expired.

2. n_i^f obtains IV from CHDR and removes one layer of encryption from the data payload:

$$\{O', IV'\} = \text{REMOVE\_LAYER}(s_i^f, IV, O) \tag{3.21}$$

3. n_i^f updates the IV field in CHDR with IV'.

4. n_i^f sends the resulting packet $\{\text{CHDR}', \text{AHDR}^{f\prime}, O'\}$ to the next node according to R_i^f.

The above procedures show that the intermediate node processing requires only symmetric-cryptography operations.

Destination processing D processes incoming data packets as the intermediate nodes. Removing the last encryption layer from the onion payload D obtains the original data payload M sent by S. Additionally, for the first data packet D retrieves AHDR^b from the payload, and stores the $\{s_D, R_0^b, \text{AHDR}^b\}$ locally so that D can retrieve AHDR^b when it wishes to send packets back to S.

Processing for the backward path Sending and processing a HORNET packet along the backward path is the same as that for the forward path, with the exception of processing involving the data payload. Because D does not possess the symmetric keys that each node on the backward path shares with S, D cannot onion-encrypt its payload. Therefore, instead of REMOVE_LAYER, D and the intermediate nodes use ADD_LAYER to process the data payload, and the source node recovers the data with REMOVE_LAYER.

3.4 Security Analysis

This section describes how HORNET defends against well-known de-anonymization attacks and meets the design goals of Section 3.1.3. We also present defenses against denial of service attacks. A taxonomy of attacks against low-latency anonymity systems, as well as formal proofs showing that HORNET satisfies the correctness, security, and integrity properties defined by Camenisch and Lysyanskaya [51] are detailed in the full version [59].

3.4.1 Passive De-anonymization

Session linkage Each session is established independently from every other session, based on fresh, randomly generated keys. Sessions are in particular not related to any long term secret or identifier of the host that creates them. Thus, two sessions from the same host are unlinkable, i.e., they are cryptographically indistinguishable from sessions of two different hosts.

Forward/backward flow correlation The forward and backward headers are derived from distinct cryptographic keys and therefore cannot be linked. Only the destination is able to correlate forward and backward traffic, and could exploit this to discover the round-trip time (RTT) between the source and itself, which is common to all low-latency anonymity systems. Sources willing to thwart such RTT-based attacks from malicious destinations could introduce a response delay for additional protection.

Packet correlation HORNET obfuscates packets at each hop. This prevents an adversary who observes packet bit patterns at two points on a path from linking packets between those two points. In addition to onion encryption, we also enforce this obfuscation by padding the header and the payload to a fixed length, thwarting packet-size-based correlation.[5] While this does not prevent the adversary from discovering that the same flow is passing his observation points using traffic analysis, it makes this process non-trivial, and allows upper-layer protocols to take additional measures to hide traffic patterns. The hop-by-hop encryption of the payload also hides the contents of the communication in transit, protecting against information leaked by upper layer protocols that can be used to correlate packets.

Path length and node position leakage HORNET protects against the leakage of a path's length and of the nodes' positions on the path (i.e., the relative distance, in hops, to the source and the destination). In the setup phase, this protection is guaranteed by Sphinx, so only the common header and FS Payload are subject to leakage (see Section 3.2.3 for the exact structure of the packets). It is straightforward to see that the common header does not contain path or position information. The FS Payload length is padded to the maximum size, and remains constant at each hop (see Algorithm 1). After adding its FS to the front of the FS Payload, each node re-encrypts the FS payload, making it infeasible for the next nodes to see how many FSes have previously been inserted.

During data transmission, neither the common header nor the data payload contain information about path length or node position, so only the AHDR (anonymous header) needs to be analyzed. The AHDR is padded to a maximum length with random bytes, and its length remains constant as it traverses the network (see Algorithm 3). The FSes contained in the AHDR are onion encrypted, as is the padding added at each hop. Thus, it is not possible to distinguish the initial random padding from the encrypted FSes, and neither of these from encrypted padding added by the nodes.

Timing for position identification A malicious node could try to learn its position on the path of a session by measuring timing delays between itself and the source (or the destination) of that session. HORNET offers two possible countermeasures. In the first, we assume that the malicious node wishes to

[5] A bandwidth-optimized alternative would be to allow two or three different payload sizes, at the cost of decreased anonymity.

measure the network delay between itself and the source. To perform such a measurement, the node must observe a packet directed to the source (i.e., on the backward path) and then observe a response packet from the source (on the forward path). However, HORNET can use asymmetric paths [100], making this attack impossible if the single node is not on both forward and backward paths.

The second countermeasure is that, even if the node is on both paths, it is still non-trivial to discover that a specific forward flow corresponds to a certain backward flow, since the forwarding segments for the two paths are independent. To link the forward and backward flows together the node would need to rely on the traffic patterns induced by the upper-layer protocols that are running on top of HORNET in that session.

3.4.2 Active De-anonymization

Session state modification The state of each node is included in an encrypted FS. During the session setup, the FSes are inserted into the FS payload, which allows the source to check the integrity of these FSes during the setup phase. During data transmission, FSes are integrity-protected as well through per-hop MACs computed by the source. In this case, each MAC protecting an FS is computed using a key contained in that FS. This construction is secure because every FS is encrypted using a PRP keyed with a secret value known only to the node that created the FS: if the FS is modified, the authentication key that the node obtains after decryption is a new pseudo-random key that the adversary cannot control. Thus, the probability of the adversary being able to forge a valid MAC is still negligible.

Path modification The two HORNET data structures that hold paths (i.e., FS payloads in the setup phase and AHDRs), use chained per-hop MACs to protect path integrity and thwart attacks like inserting new nodes, changing the order of nodes, or splicing two paths. The source can check such chained per-hop MACs to detect the modifications in the FS payload before using the modified FS payload to construct AHDRs, and similarly intermediate nodes can detect modifications to AHDRs and drop the altered packets. These protections guarantee path information integrity as stated in Section 3.1.3.

Replay attacks Replaying packets can facilitate some types of confirmation attacks [151]. For example, an adversary can replay packets with a pre-selected pattern and have a colluding node identify those packets downstream. HORNET offers replay protection through session expiration; replayed packets whose sessions have expired are immediately dropped. Replay of packets whose sessions are not yet expired is possible, but such malicious behavior can be detected by the end hosts. Storing counters at the end hosts and including them in the payload ensures that replays are recognizable. The risk of detection

helps deter an adversary from using replays to conduct mass surveillance. Furthermore, volunteers can monitor the network, to detect malicious activity and potentially identify which nodes or group of nodes are likely to be misbehaving. Honest ASes could control their own nodes as part of an intrusion detection system.

3.4.3 Payload Protection

Payload secrecy Data packet payloads are wrapped into one layer of encryption using the key shared between the source and the destination, both for packets sent by the source on the forward and for packets sent by the destination on the backward path (see Section 3.3.4.4). Assuming that the cryptographic primitives used are secure, the confidentiality of the payload is guaranteed as long as the destination is honest. In Section 2.3 we discuss the guarantees for perfect forward secrecy for the data payload.

Payload tagging or tampering HORNET does not use per-hop MACs on the payload of data packets for efficiency and because the destination would not be able to create such MACs for the packets it sends (since the session keys of the nodes are known only to the source). The lack of integrity protection allows an adversary to tag payloads. Admittedly, the use of tagging, especially in conjunction with replay attacks, allows the adversary to improve the effectiveness of confirmation attacks. However, end-to-end MACs protect the integrity of the data, making such attacks (at a large scale) detectable by the end hosts.

3.4.4 Denial-of-Service (DoS) Resilience

Computational DoS The use of asymmetric cryptography in the setup phase makes HORNET vulnerable to computational DoS attacks, where adversaries can attempt to deplete a victim node's computation capability by initiating a large number of sessions through this node. To mitigate this attack, HORNET nodes can require each client that initiates a session to solve a cryptographic puzzle [75] to defend against attackers with limited computation power. Alternatively, ISPs offering HORNET as a service can selectively allow connections from customers paying for the anonymity service.

State-based DoS HORNET is not vulnerable to attacks where adversaries maintain a large number of active sessions through a victim node. One of HORNET's key features is that all state is carried within packets, thus no per-session memory is required on nodes or rendezvous points.

3.4.5 Topology-based Analysis

Unlike onion routing protocols that use global re-routing through overlay networks (e.g., Tor [79] and I2P [196]), HORNET uses short paths created by the underlying network architecture to reduce latency, and is therefore bound by the network's physical interconnection and ISP relationships. This is an unavoidable constraint for onion routing protocols built into the network layer [105, 156]. Thus, knowledge of the network topology enables an adversary to reduce the number of possible sources (and destinations) of a flow by only looking at the previous (and next) hop of that flow. For example, in Figure 3.3, assume that AS0 is controlled by a passive adversary. The topology indicates that any packet received from AS1 must have originated from a source located at one of {AS1, AS2, AS3, AS4, AS5}.

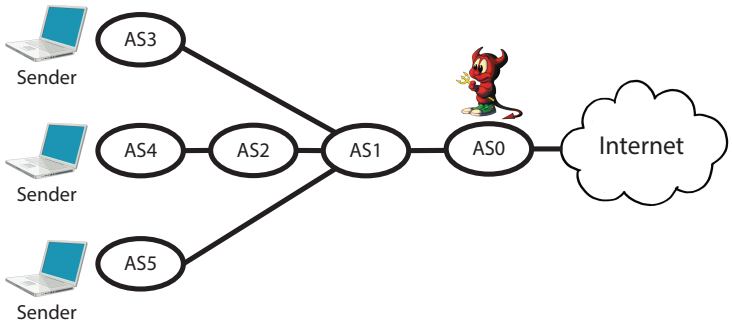
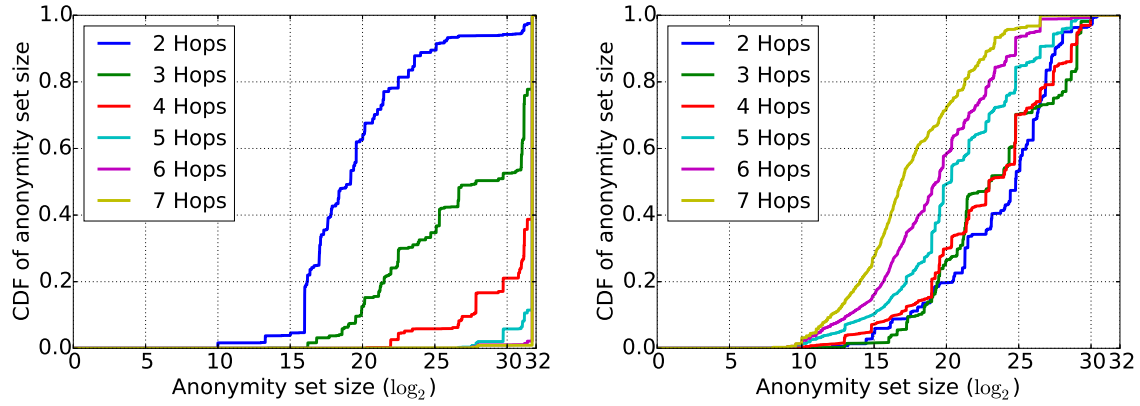


Figure 3.3: An example AS-level topology with an adversarial AS (AS0).

We evaluate the information leakage due to the above topology constraints in the scenario where a single AS is compromised. We derive AS-level paths from iPlane trace-route data [12], and use AS-level topology data from CAIDA [121]. For each AS on each path we assume that the AS is compromised and receives packets from a victim end host through that path. We compute the end host's anonymity set size learned by the adversary according to the topology. For instance, in Figure 3.3, if AS0 is compromised and receives from AS1 packets originally sent by a user in AS4, we compute the size of the anonymity set composed of all the ASes that can establish valley-free paths traversing the link from AS1 to AS0. In this example, the anonymity set size would be the sum of the sizes of AS1, AS2, AS3, AS4, and AS5.

Similar to Hsiao et al. [105], we use the number of IPv4 addresses to estimate the size of each AS. Figure 3.4(a) plots the CDF of the anonymity set size for different distances (in number of AS hops) between the adversary and the victim end host. For adversarial ASes that are 4 hops away, the anonymity set size is larger than 2^{31} in 80% of the cases. Note that the maximum anonymity set size is 2^{32} in our analysis, because we consider only IPv4 addresses.



(a) Without path knowledge

(b) With path knowledge

Figure 3.4: Evaluation fo HORNET anonymity set size. a) CDF of anonymity-set size when a position-agnostic AS on path is adversarial. "Hops" indicates the number of ASes between the adversarial AS and the victim end host. For example, the point (25, 0.4) on the line "3 hops" means that the anonymity set size is smaller than 2^{25} in 40% of cases when the end host is 3 hops away from the adversarial AS. b) CDF of anonymity-set size when an adversarial AS knows its own position on the path. For both Figures a) and b), the maximum size of an end host's anonymity set is 2^{32} because we consider the IPv4 address space. Therefore, the ideal case for an end host is that the anonymity set size is 2^{32} with probability equal to 1.

Implications of path knowledge Knowledge about the path, including the total length of the path and an adversarial node's position on the path, significantly downgrades the anonymity of end hosts. Considering again Figure 3.3, if the adversary controlling AS0 sees a packet incoming from AS1 and knows that it is 4 hops away from the source host, he learns that the source host is in AS4. Compared with the previous case, we see that the anonymity set size is strongly reduced.

We quantify additional information leakage in the same setting as the previous evaluation. Figure 3.4(b) represents the CDFs of the anonymity set sizes of end hosts according to the distance to the compromised AS. The anonymity set sizes are below 2^{28} in 90% of the cases when the adversarial ASes are 4 hops away, with an average size of 2^{23}. This average size decreases to 2^{17} for the cases where the adversarial ASes are 7 hops away from the target hosts.

Previous path-based anonymity systems designed for the network layer either fail to hide knowledge about the path [156] or only partially obscure the information [105]. In comparison, HORNET protects both the path length and the position of each node on the path, which significantly increases the anonymity-set size.

3.5 Evaluation

We implemented the HORNET router logic in an Intel software router using the Data Plane Development Kit (DPDK) [8]. To our knowledge, no other anonymity protocols have been implemented in a router SDK. We also implemented the HORNET client in Python. Furthermore, we assembled a custom crypto library based on the Intel AESNI cryptographic library [10], the curve25519-donna library [7], and the PolarSSL libraries [17]. We use IP forwarding in DPDK as our performance baseline. For comparison, we implemented the data forwarding logic from Sphinx, LAP, Dovetail, and Tor using DPDK and our cryptographic library.

Fairly comparing the performance of anonymity systems at the application layer with those that operate at the network layer is challenging. To avoid penalizing Tor with additional propagation delay caused by longer paths and processing delay from the kernel's network stack, we implemented Tor at the network layer (as suggested by Liu et al. [120]). Tor's design requires relay nodes to perform SSL/TLS and transport control. SSL/TLS between neighboring relays at the application layer maps to link encryption between neighboring nodes at the network layer, which we consider orthogonal but complementary to HORNET. Hence, for fair comparison, we implemented the network-layer Tor without SSL/TLS or transport control logic. Throughout our evaluation we refer to this implementation of Tor as L3 Tor.

Our testbed contains an Intel software router connected to a Spirent TestCenter packet generator and analyzer [20]. The software router runs DPDK 1.7.1 and is equipped with an Intel Xeon E5-2680 processor (2.70 GHz, 2 sockets, 16 logical cores/socket), 64 GB DRAM, and 3 Intel 82599ES 40 Gb/s network cards (each with 4 10 Gb/s ports). We configured DPDK to use 2 receiving queues for each port with 1 adjacent logical core per queue.

3.5.1 Data Forwarding Performance

Forwarding latency We measure the CPU cycles consumed to forward a data packet in all schemes. Figure 5.5 shows the average latency (with error bars) to process and forward a single data packet in all schemes (except Sphinx[6]) when payload sizes vary. We observe that HORNET, even with onion encryption/decryption over the entire payload and extensive header manipulation, is only 5% slower than LAP and Dovetail for small payloads (64 bytes). For large payloads (1200 bytes[7]), HORNET is 71% slower

[6]We omit Sphinx from the comparison for better readability. In our experiments, processing a Sphinx packet takes more than 640K cycles due to asymmetric cryptographic operations. This is 3 orders of magnitude slower than that of HORNET, L3 Tor, LAP, and Dovetail.

[7]Because LAP, Dovetail, and HORNET all have large packet headers of 300+ bytes, we limit the largest payload in our experiments to be 1200 bytes.

Scheme	Header Length	Sample Length (Bytes)
LAP	$12 + 2s \cdot r$	236
Dovetail	$12 + s \cdot r$	124
Sphinx	$32 + (2r + 2)s$	296
Tor	$3 + 11 \cdot r$	80
HORNET	$8 + 3r \cdot s$	344

Table 3.2: Comparison between the length of different packet header formats in bytes. s is the length of symmetric elements and r is the maximum AS path length. For the sample length, we select $s = 16$ Bytes and $r = 7$. Analysis of iPlane paths shows that more than 99% of all paths have fewer than 7 AS hops.

(about 400 nanoseconds slower per packet when using a single core) than LAP and Dovetail. However, the additional processing overhead enables stronger security guarantees.

Header overhead As a result of carrying anonymous session state (specifically cryptographic keys) within packet headers, HORNET headers are larger than Sphinx, L3 Tor, LAP, and Dovetail headers (see Table 3.2). While larger headers reduce net throughput (i.e., goodput), this tradeoff appears acceptable: compared to L3 Tor, no state is required at relay nodes, enabling scalability; compared to Sphinx, data processing speed is higher; compared to LAP and Dovetail, HORNET provides stronger security properties.

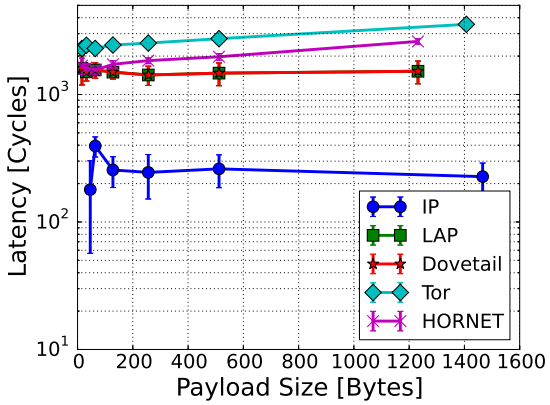
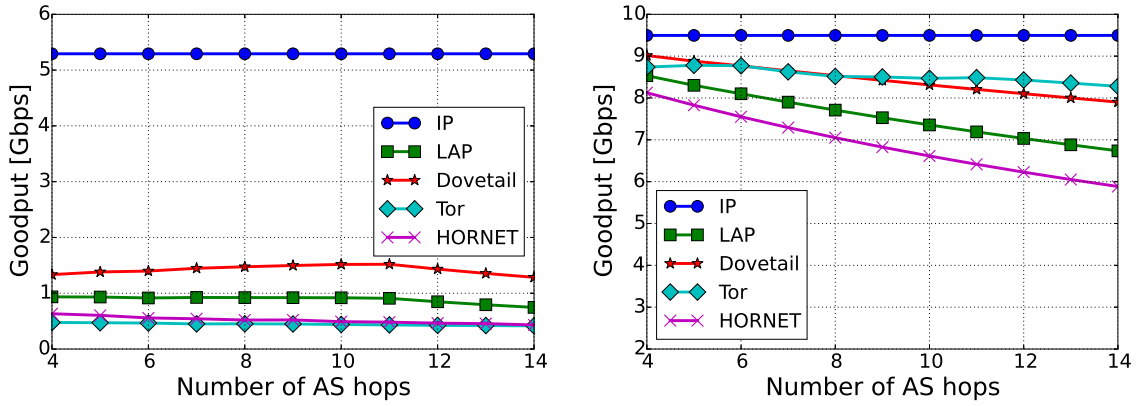


Figure 3.5: Per-node data HORNET forwarding latency on a 10 Gbps link. Lower is better.

Goodput We further compare all the schemes by goodput, which excludes the header overhead from total throughput. Goodput is a comprehensive metric to evaluate both the packet processing speed and protocol overhead. For example, a scheme where headers take up a large proportion of packets yields only low goodput. On the other hand, a scheme with low processing speed also results in poor goodput.



(a) 40 Byte payloads

(b) 1024 Byte payloads

Figure 3.6: Evaluation HORNET goodput. a) Data forwarding goodput on a 10 Gbps link for small packets (40 Byte payloads); b) Data forwarding goodput large packets (1024 Byte payloads). Higher is better.

Figure 5.6(a) and Figure 5.6(b) demonstrate the goodput of all schemes (except Sphinx[8]) on a 10 Gb/s link when varying the number of hops r, with 40-byte and 1024-byte payloads, respectively. Larger r means larger header sizes, which reduces the resulting goodput.

When the payload size is small, the goodput of all protocols remains stable. This is due to the fact that no scheme can saturate the link, and accordingly the goodput differences between the three schemes mainly reflect the different processing latencies among them. Consequently, L3 Tor's and HORNET's goodput is 32% less than that of LAP and Dovetail. On the other hand, when the payload size is large, all schemes except Sphinx can saturate the 10 Gb/s link. HORNET can reach 87% of LAP's goodput while providing stronger security guarantees.

3.5.2 Max Throughput on a Single Router

To investigate how our implementation scales with respect to the number of CPU cores, we use all 12 ports on the software router, generating HORNET data packets at 10 Gb/s on each port. Each packet contains a 7 AS-hop header and a payload of 512 bytes, and is distributed uniformly among the working ports. We monitor the aggregate throughput on the software router.

The maximal aggregate throughput of HORNET forwarding in our software router is 93.5 Gb/s, which is comparable to today's switching capacity of a commercial edge router [5]. When the number of cores ranges from 1 to 4, our HORNET implementation can achieve full line rate (i.e., 10 Gb/s per port). As the

[8]Sphinx's goodput is less than 10 Mb/s in both cases because of its large packet headers and asymmetric cryptography for packet processing.

number of cores increases to 5 and above, each additional port adds an extra 6.8Gb/s.

3.5.3 Session Setup Performance

We evaluate the latency introduced by processing setup packets on each border router. Similar to measuring the latency of data forwarding, we also instrument the code to measure CPU cycles consumed to process packets in the session setup phase. Table 3.3 lists the average per-node latency for processing the two setup packets in HORNET's session setup phase. Due to a Diffie-Hellman key exchange, processing the two setup packets in the session setup phase increases processing latency (by about 240μs) compared to data packet processing. However, HORNET must only incur this latency once per session.

Packet	Latency (K cycles)	Latency (μs)
P❶	661.95 $\pm$ 30.35	245.17 $\pm$ 11.24
P❷	655.85 $\pm$ 34.03	242.91 $\pm$ 12.60

Table 3.3: Per-node latency to process HORNET session setup packets with standard errors.

Chapter 4

PHI: Path-Hidden Lightweight Anonymity Protocol at the Network Layer

In the previous chapter, HORNET improves the anonymity of the lightweight anonymous communication systems, i.e., LAP and Dovetail, at the cost of additional latency, computation, and bandwidth overhead. In this chapter, we propose a Path-HIdden lightweight anonymity protocol, named PHI, that improves anonymity over LAP and Dovetail and is equally efficient.

PHI improves on LAP and Dovetail by introducing three new techniques. First, PHI places nodes' state in a pseudo-random order in a packet header to conceal information about node positions. Second, PHI leverages a back-off path construction method to eliminate the need for a source to fully control the path traversed. Third, PHI prevents session hijacking attacks by binding payload encryption to paths.

In this chapter, we make the following contributions:

1. We identify two attacks that reduce sizes of anonymity sets in LAP and Dovetail. In particular, we model and analyze the path information leakage when LAP and Dovetail intentionally obscure path information by using *variable-size segments* (see Section 4.1.1). In comparison, HORNET only shows that LAP and Dovetail leak path information without such protection mechanism.

2. We propose a path-hidden header format that is more efficient than the ones used by onion routing protocols.

3. We present a new approach to establish an end-to-end path for a source node with no control over the path traversed.

4. We design the Path-HIdden lightweight anonymity protocol (PHI), an efficient network layer protocol that provides stronger anonymity properties than LAP and Dovetail with the same level of

efficiency.

5. We evaluate PHI's security and performance. Evaluation results confirm that PHI's performance is comparable to, or more efficient than LAP and Dovetail, while expanding the sizes of anonymity sets.

The contents of this chapter are closely related to the paper published at PETS'17 [60].

4.1 Challenges of the Existing Lightweight Anonymity Protocols

4.1.1 Topology-based Attacks and Variable-size Segment

Unlike a node in an overlay-based anonymity system that can forward packets to any other node, an AS can only forward packets according to its physical connections and business contracts with its neighbors. Therefore, network-layer anonymity protocols are inherently subject to so called "topology-based attacks". With publicly-available network topology information, a compromised node receiving a single packet from a victim can narrow down the victim's anonymity set. For example, in Figure 4.1, if we assume that AS2 is AS3's customer, by knowing the topology and receiving a packet from AS2, AS3 can conclude that the source node must be within {AS0, AS1, AS2, AS4, AS5}, which also forms the anonymity set. If the adversarial AS further discovers its AS-level distance from the source's AS, it can reduce the size of the victim's anonymity set. In Figure 4.1, if AS2 is AS3's customer and AS3 uncovers that the source of a packet from AS2 is 3 hops away, AS3 can infer that the source's anonymity set only contains AS0, which significantly harms the source's anonymity.

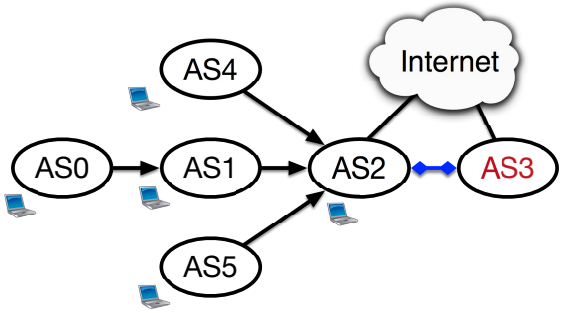


Figure 4.1: **Topology-based attacks**. AS3 is a compromised AS and receives a packet form AS2 and conducts topology-based attacks to de-anonymize the packet's source node. The arrow from an AS points to its provider and the relationship between AS2 and AS3 can be peer-to-peer, customer-to-provider, and provider-to-customer.

In both LAP and Dovetail, path segments are successively appended to a header by each on-path node, and each header tells the forwarding router which path segment to process. As a result, even a single

compromised router can determine the number of hops between the end host and itself by counting path segments in the header. As demonstrated by Chen et al. [57], such an attack can successfully reduce the anonymity-set size of the source and destination. HORNET, on the other hand, avoids leaking such information by onion encrypting and shifting path segments in a header.

To defend against such attacks, LAP proposes using *Variable-Size Segments* (VSS) to hide path information. With VSS, each path segment is padded to a random size of multiple blocks chosen independently by on-path ASes. However, VSS doubles or triples the packet-header size, adding to bandwidth overhead. If the maximum number of segment blocks is A, the resulting VSS path in a packet header is A times larger than a path without VSS. In the following, we call A "path-size amplification factor", or simply "amplification factor".

4.1.2 Identified Challenges of Lightweight Protocols

We identify two attack vectors to existing lightweight protocols that can further reduce anonymity-set sizes of end hosts.

4.1.2.1 Path Information Leakage

Although VSS helps obscure the number of path segments inserted in a packet header, it cannot fully hide the information. In fact, an on-path adversary can still determine the probability distribution of its position on the path and the path length. With the knowledge of the distribution, the adversary can increase the probability of identifying a victim and thus reduce *equivalent* anonymity-set size. For example, in Figure 4.1, if each AS can output a path segment of either 1 or 2 basic blocks, and AS3 observes that there are 6 basic blocks before its own path segment, AS3 can derive that the source node must reside in AS0. We will mathematically model the path information leakage of VSS in Section 4.5.

4.1.2.2 Session Hijacking Attacks

In both LAP and Dovetail, payload encryption is essential to prevent adversaries from deanonymizing either end by scrutinizing identity information in packet payloads. A common practice is to use DH key exchange to negotiate a shared symmetric key for encrypting subsequent payloads in a session. In order to thwart an adversary in hijacking the key exchange protocol by launching Man-in-the-Middle (MitM) attacks, both parties should verify the public key of the other end.

Nevertheless, when sender anonymity is at stake, end hosts can only conduct one-way authentication, i.e., a source verifies the public key of its destination but not vice versa. Consequently, an on-path adver-

sarial node is able to hijack the key exchange session by replacing the source's public key with its own. Because the destination cannot verify that the public key is from an anonymous source or from an adversarial node, the adversary can communicate with the destination in place of the source, and compromise the destination's identity.

Note that in LAP an on-path adversarial node can directly obtain the destination from setup packets without launching session hijacking attacks. Dovetail, improving over LAP by using a helper node to conceal the destination's identity, is susceptible to the hijacking attack described above. A first-hop node, who knows the source by its advantageous position, can leverage such an attack to deanonymize the destination in the following way. The adversary waits until a source node establishes an end-to-end path using Dovetail's session setup, during which process the adversary can obtain the created path as well. The adversary then uses the intercepted path to communicate with the destination and sets up a secure channel by using the DH key exchange protocol. Because the destination cannot verify that the public key of the other end, it cannot tell whether it communicates with the actual source or an adversarial node. With this attack, the first-hop node can potentially compromise the destination's identity, defeating Dovetail's security goal.

4.2 Problem Definition

4.2.1 Network Model

We regard ASes as the network entity to route packets. Each AS maintains a set of interfaces through which all traversing traffic enters and exits the AS. Depending on the inter-domain routing protocol agreed upon by all ASes, an AS can route an incoming packet by either locally-stored forwarding information (e.g., BGP [154]) or routing information within the packet itself (e.g., NIRA [195], Pathlet [92], and SCION [198]). We also consider a model with loose AS boundaries. In particular, in Pathlet [92], an AS is further divided into virtual nodes (vnodes), each of which serves as an independent principal to forward packets. In this chapter, we use the generic term *node* to denote an AS or a vnode in the case of Pathlet. We consider the fact that ASes in the inter-domain network follow various network policies, and our anonymity protocol should be versatile in supporting existing network policies.

For anonymity purposes, we assume that an AS upgrades its border routers to support necessary symmetric cryptography such as AES. Each AS also keeps a local secret key and updates it periodically. All the anonymity-supporting routers should share the secret key and synchronize the key updates.

4.2.2 Communication Model

A source can anonymously communicate with a destination through a session, which is composed of traffic that shares cryptographic state. Each session is divided into two phases: a setup phase and a forwarding phase.

The setup phase helps the source and destination to establish necessary state to communicate anonymously. First, the source creates a setup request based on the intended destination. Then the source sends the initial packet, called setup request, through the path. Routers of on-path ASes insert necessary forwarding state into the setup packet. When the setup request reaches the destination, the destination constructs a reply based on the request and sends it back to the source. As the reply reaches the source, the source can extract all cryptographic state required to communicate in the session.

The forwarding phase enables high-speed packet forwarding. The source and destination create packet headers for data packets belonging to that session. Processing the headers allows a router to determine the next router without learning information beyond the router's previous and next-hop nodes.

4.2.3 Threat Model

We target a curious but cautious adversary: the adversary aims to link an action (e.g., visiting a webpage) to an identity in a given anonymity set but wants to avoid detection. We allow the adversary to compromise any single AS, or any destination host in the network. By compromising an entity, the adversary obtains all the entity's cryptographic keys. The adversary can perform passive deep packet inspection or active traffic manipulation, such as dropping, injecting, and modifying packets that traverse a compromised entity. Furthermore, we assume that the adversary possesses full knowledge about the network topology. However, we do not consider an adversary that can perform passive or active traffic analysis. Our adversary model is in line with Dovetail and removes the assumption about trusting the first-hop node in LAP.

4.2.4 PHI's Security and Performance Goals

PHI achieves the following security goals:

- *Sender/sender-receiver anonymity*. PHI offers two types of anonymity properties: sender anonymity and sender-receiver anonymity, as defined by Pfitzmann and Köhntopp [144]. As a network-layer anonymity protocol, PHI considers an end host to be *anonymous* if its network location cannot be distinguished from that of a set of other hosts, called the anonymity set. However, because the first-

hop AS is directly connected to the source node, PHI only offers *relationship anonymity* [144] against the first-hop adversary.

- *Session unlinkability*. Traffic from different sessions cannot be linked together by an adversary to de-anonymize end hosts.
- *Path-information confidentiality/authenticity*. An on-path compromised node cannot uncover the total number of nodes on the path or its distance from end hosts on either side. In addition, an adversary cannot modify an existing path or fake new paths so that resulting paths still traverse the network.

PHI also aims to offer a series of performance properties:

- *Low bandwidth overhead*. PHI adds a small header to each data packet.
- *Low latency*. Processing a PHI packet only incurs low additional latency.
- *Scalability*. PHI nodes maintain no per-flow state.

4.3 Design

PHI adopts two design features that are common among network-layer anonymity systems. First, for high-speed data-packet forwarding, it uses only symmetric cryptography for packet forwarding to achieve low end-to-end latency and high throughput. Second, to scale to large traffic volumes, PHI packet headers carry state required by routers to forward the packets, relieving routers from storing per-session state that can overflow their limited memory.

PHI offers stronger anonymity than Dovetail but only requires the same level of overhead, and is compatible with legacy network architectures. In particular, PHI defeats the identified attacks of existing lightweight protocols, i.e., that headers leak information about node positions and that they are vulnerable to session hijacking attacks.

We now describe the core ideas introduced in this chapter. PHI's packet header design randomizes path-segment positions to prevent leaking topological information, and enlarges the expected anonymity set compared to VSS. (Section 4.3.2). Next, PHI uses a helper node to hide the destination from the nodes close to the source and allows the network itself to select the helper node using a back-off method (Section 4.3.3). Therefore, PHI is compatible with network architectures where dynamic global topology information is inaccessible, e.g., the Internet, not limiting itself to Pathlet. Lastly, during the path setup phase, PHI binds the source's public key to the established path, thwarting an on-path node from launching session hijacking attacks (Section 4.3.4).

4.3.1 PHI High-level Overview

Figure 4.2 shows an example of PHI path setup phase. Suppose a source node S wants to anonymously communicate with D using PHI. It starts by picking a third node M to establish a full path from S to D. In this dissertation, we call M the *helper node*. M can be either a service operated by an ISP or a server voluntarily run by another user. During the entire process, M knows D's network location but has no knowledge about S's. In addition, M only forwards on average fewer than 2 setup packets for each session.

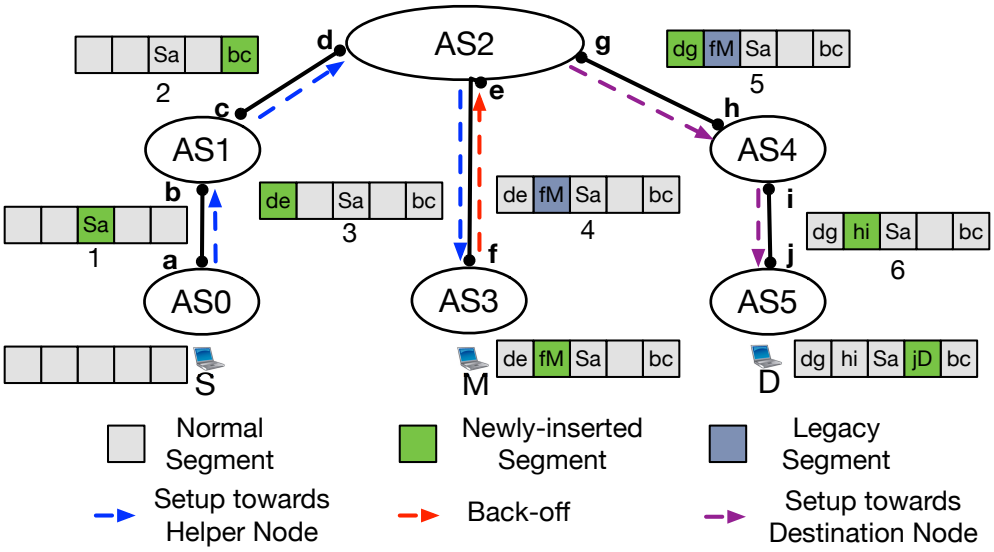


Figure 4.2: An example of PHI session setup.

First, S encrypts D's address using M's public key and creates a path to M. Based on M's address, each AS on the path between S and M independently decides how to forward the packet, encrypts the decision using a local secret, and places the resulting ciphertext in a pseudo-random position in the header determined by the session's identifier and the AS's local secret. In Figure 4.2, the full path between S and M is {AS0: $S \rightarrow a$, AS1: $b \rightarrow c$, AS2: $d \rightarrow e$, AS3: $f \rightarrow M$}. Each AS places its routing decision in a pseudo-random position in the header. For example, AS0 embeds $S \rightarrow a$ in the 3rd position in the header, AS1 inserts $b \rightarrow c$ into the 5th position, and etc. Because each AS independently selects a random position for its routing decision, it is possible that their selections are in conflict. We will address this issue in detail in Section 4.3.2.

Inserted routing decision is indistinguishable from empty slots. Initially, the source generates random bits to fill the empty header. Each AS encrypts its routing decision with a local secret key before inserting routing decision into the header.

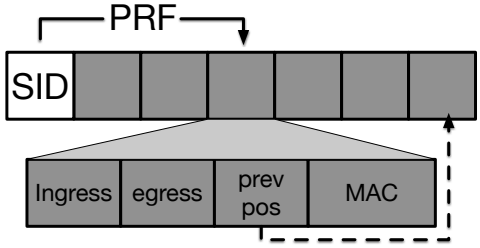


Figure 4.3: Randomize a path segment's position in a PHI header.

Once the setup packet reaches M, M decrypts D's address and initiates a back-off process that helps S find a midway node, e.g., AS2 in Figure 4.2. The midway node is responsible for forwarding all subsequent data packets in the forwarding phase between S and D. M uses the header received in the setup packets to route packets in the reverse direction. For instance, AS3 retrieves its routing decision $f \rightarrow M$ and forwards the back-off packet to AS2 through interface f.

The back-off process stops when an AS W on the path from S to M is willing to forward the packet to D through an interface that is different from the one where the first setup packet is received. We name this AS W the *midway node*. In Figure 4.2, AS2 can forward a packet to D through interface g, which is different from the interface d where the first setup packet is received.

W continues the setup process until the setup packet reaches D. D can retrieve the bi-directional end-to-end path between S and D from the setup packet's header. The rest of the setup phase and the data forwarding phase is essentially equivalent to LAP and Dovetail. D sends the created path back to S using the path in the reverse direction, and S and D can communicate anonymously using the newly established path.

4.3.2 Randomizing Path-segment Positions

The key idea behind hiding path information in PHI is to randomize the position of a node's segment in packet headers. As shown in Figure 4.3, a node n_i computes its segment's position in the header by using a Pseudo-Random Function (PRF) with its local secret k_i^{pos} as the key and the session's id sid as the input.

$$pos = \text{PRF}(k_i^{pos}; sid) \tag{4.1}$$

In addition, the node can find its previous-hop node's segment using the "previous hop" field in the packet header, denoted as pos_{prev}.

If the packet is a setup request packet, the node n_i computes its path segment S_i using the routing information R_i, a local encryption key k_i^{enc}, a local MAC key k_i^{mac}, and a MAC M_{i-1} in the previous

node's segment S_{i-1} as follows:

$$E_i = \text{ENC}(k_i^{enc}; R_i \parallel pos_{prev} \parallel flags) \tag{4.2}$$

$$M_i = \text{MAC}(k_i^{mac}; E_i \parallel M_{i-1}) \tag{4.3}$$

$$S_i = E_i \parallel M_i \tag{4.4}$$

Here, the actual form of R_i depends on the network architecture that PHI is based on. For instance, it can take the form of (*input link*, *output link*). n_i then replaces the current path segment in the packet request with the newly generated S_i.

If the packet is a data packet, n_i retrieves its path segment, decrypts the segment to find the routing information, and verifies the integrity of its path segment in the inverse process of Equations 4.2 to 4.4.

4.3.2.1 Resolving Collisions

For packet setup requests, because each node independently computes its path segment's position, it is possible that the position in the packet request is already occupied by segment of another node n_c. A collision in picking segment positions causes information loss for n_c's path segment. When the source uses the resulting packet header with collision to forward packets, the MAC verification will fail at n_c.

To avoid using a packet with collided path segments, the source sends more than one setup request packets to increase its success rate. Let the total number of on-path nodes be r, the maximum path segments in the header be m. To reach a success rate of c, the source can send N setup requests all together to satisfy the following condition:

$$1 - \left(1 - \frac{P_m^r}{m^r}\right)^N \geq c \tag{4.5}$$

where P_m^r computes the number of r-permutations of a set with n element.

Resolving local collisions . Because to each node, the position of its previous-hop node's segment is revealed by the field pos_{prev}, the node can further rule out one type of collision: its own segment's position collides with that of its previous node's segment. If such a collision happens, a node itself can resolve the collision without recourse to other nodes or end hosts. The node computes a new position as follows:

$$pos_{new} = PRF(s; sid \oplus ctr) \tag{4.6}$$

where ctr is the first value satisfying $pos_{new} \neq pos_{prev}$ starting from 1.

Resolving the local collision in the above manner poses a new challenge for retrieving the segment for a node. Since the node attempting to retrieve its segment from a packet header has no prior knowledge

about where its previous-hop node's locates, it may end up with choosing the default position with ctr = 0. However, we remark that the node can detect this scenario by verifying the MAC contained in the segment. A failure indicates that there must be a local collision when setting up the packet header and the node should increment the ctr to locate a new position to retrieve its segment.

The resulting number of setup requests, N, satisfies:

$$1 - \left(1 - \frac{P_m^r}{m(m-1)^{r-1}}\right)^N \leq c \tag{4.7}$$

4.3.2.2 Probabilistic Header Sizes

The above method runs into a dilemma when the path length r is close to the maximal path segments that a header can contain m. For example, as Section 4.6.1 shows, when $m = 12$, the required number of setup packets to reach 90% success rate amounts to 4. We can increase m to reduce the required number of setup packets. When $m = 21$, the source only needs 3 setup requests to guarantee 90% success rate. However, with a larger m comes larger packet header for every data packet, which proportionally increases the bandwidth overhead.

However, we observe that the probability that a long path exists between a random pair of source and destination is low. According to our experiment in Section 4.6.1, the probability of a path that is longer than 7 AS hops is below 0.01%. As a result, we can leverage a probabilistic header size to mitigate the dilemma between the number of setup packets and the bandwidth overhead.

When the source finds the total path length r is above 7, the source always selects a large value for m, e.g., 48, to keep the number of sent packets small. When r is below 7, the source still chooses a large m with probability P. The source selects a small m for the rest of the paths. In the scenario where $P = 0$, the algorithm is equivalent to using separate header sizes for long and short paths, which leaks path length information by packet header sizes. The source can select larger values for P to add stronger protection to conceal the fact that a long path is used.

To estimate path lengths r, the source can download an AS-level topology, e.g., from CAIDA [121], and run a local simulator to calculate the path length. The resulting path length is in turn an input into the algorithm to determine the size of the path. Note that a wrong estimate of r due to inaccurate simulation can potentially increase the collision rate and increase setup latency when the estimated r is smaller than the real path length.

4.3.3 Back-off Path Setup

To hide destinations' addresses from ASes that are close to the source, PHI uses helper nodes as indirection for source nodes to reach their destinations. Helper nodes can either be end hosts or ISPs that provide PHI indirection service. The information about which nodes serve as helper nodes can be distributed to end hosts or retrieved by end hosts without compromising anonymity of communication that happen afterwards.

PHI's assumptions are weaker than Dovetail: source nodes in PHI do not have to possess control over the path that packets traverse. This relaxed assumption allows PHI to operate on architectures like the Internet where a source lacks control over packets' paths.

In this subsection, we discuss the back-off path setup employed by PHI to support the relaxed assumption. We then analyze the resulting anonymity-set size of the proposed technique.

4.3.3.1 Path Setup Using Helper Nodes

Figure 4.4 shows how to establish an end-to-end path using a helper node in PHI. To construct a PHI path to reach the destination D, a source node S from AS0 selects a helper node M in AS4 and establishes a first half path towards M. Like LAP, S only places M's address into the setup request, so that each on-path AS can determine how to approach M. On-path nodes AS1 to AS4 follow the algorithm described in Section 4.3.2 to determine the positions of each individual path segment and insert them into the setup request (see Section 4.4).

In addition, S also encrypts D's address by using ECDH with M's public key. When M receives the setup request packet, it can obtain D's address and continue to establish the second half path to D.

Nevertheless, simply concatenating the first and second half paths to form the full path is undesirable. On one hand, M may not want to, or be able to, forward all traffic between S and D. On the other hand, the resulting long path incurs additional communication latency.

In PHI, M, on behalf of S, finds a node W on the first half path, called "midway node", that forwards data packets to D. In Figure 4.4, AS2 serves as the midway nodes that bridge the first half and second half paths together. Instead of traversing AS3 and AS4 twice, data packets can be directly forwarded towards D by AS4.

M uses a "back-off" technique to find the midway node with the network's help, while preserving each node's local policy. M sends a midway request packet (see Section 4.4) in the reverse direction of the first half path towards S. M also embeds D's address in the midway request packet. Each on-path node checks its previous hop and D's address, and decides whether it is the midway node based on its local

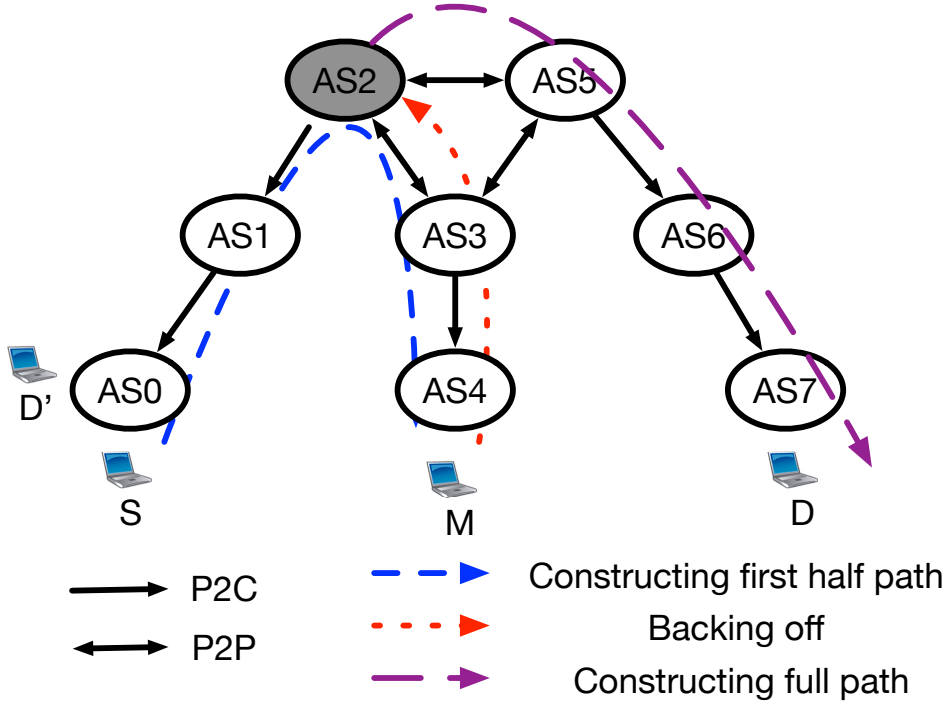


Figure 4.4: Back-off path setup. The arrow standing for P2C relationship always points from the provider to its customer. The shaded node is the midway node. We call the path between S and M the first "half path" and the one between M and D the second "half path".

policy. For example, AS4 cannot be the midway node because of the valley-freeness policy: it will not forward a packet from its provider AS4 back to AS4. In comparison, AS2 is the first node that can forward packets from AS1 to AS7 without breaking valley-freeness. Therefore, AS3 becomes the midway nodes. Once chosen, the midway node sends a path request to the D based on the received midway request.

PHI's header format inherently supports the above back-off technique because of its "automatic recycling" feature. When receiving a midway request packet containing the first half path, an on-path node can retrieve its previous path segment, verify its integrity, and decrypt information including the previous hop node. If the node decides not to become the midway node, its path segment in the header is automatically recycled by other nodes: another node can insert its path segment in the same position as an old segment to overwrite the legacy one.

Automatic recycling keeps header sizes small. As demonstrated in Section 4.3.2, to avoid setup failure caused by path-segment collisions, the header size increases exponentially with respect to the length of the path contained. With path-segment recycling, S only needs to consider the length of the longer path between the path from S to M and the one from M to D.

Node group	$\lvert\mathcal{A}_S\rvert$	$\lvert\mathcal{A}_D\rvert$
n between S and W	$1 \leq \lvert\mathcal{A}_S(n)\rvert \leq \lvert\mathcal{N}\rvert$	$\lvert\mathcal{N}\rvert$
W	$1 \leq \lvert\mathcal{A}_S(W)\rvert \leq \lvert\mathcal{N}\rvert$	1
n between W and M	$\lvert\mathcal{A}_S(W)\rvert \leq \lvert\mathcal{A}_S(n)\rvert \leq \lvert\mathcal{N}\rvert$	1
M	$\lvert\mathcal{N}\rvert$	1
n between W and D	$\lvert\mathcal{A}_S(W)\rvert \leq \lvert\mathcal{A}_S(n)\rvert \leq \lvert\mathcal{N}\rvert$	1

Table 4.1: Anonymity-set sizes observed by each group of nodes, if compromised.

4.3.3.2 Maximizing Anonymity-set Size

To quantify each session's anonymity, we consider anonymity sets of source-destination pairs. Let $\mathcal{N}$ be the maximum anonymity set for source or destination that contains all possible end hosts, $\mathcal{A}(n)$, $n \in \mathcal{N}$ be the anonymity set of source-destination pairs that a compromised node n_i observes. If we further denote the anonymity set of the source node that an adversarial n observes $\mathcal{A}_S(n)$ and the anonymity set of the destination node that n observes $\mathcal{A}_D(n)$, we have the following relationship:

$$|\mathcal{A}(n)| = |\mathcal{A}_S(n)| \times |\mathcal{A}_D(n)| \tag{4.8}$$

where $|\cdot|$ is the size of a set.

We show that the midway node W, if compromised, can observe the smallest anonymity set. Consider $\mathcal{A}_S(n)$ for the first half path. It increases monotonically for the nodes in the order from S to M. For instance, in Figure 4.4, $\mathcal{A}_S(\text{AS3})$ is larger than $\mathcal{A}_S(\text{AS2})$, which in turn is larger than $\mathcal{A}_S(\text{AS1})$. For nodes in the second half path, $\mathcal{A}_S(n)$ is always $\mathcal{N}$. As for $\mathcal{A}_D(n)$, the resulting anonymity set is $\mathcal{N}$ for nodes before the midway point, but $\mathcal{A}_D(n)$ for nodes after M is $\{D\}$ because these nodes know D's address from the midway request and the second path request. Table 4.1 summarizes the anonymity-set size for each group of nodes.

Hence, we can use $\mathcal{A}(W)$ as a metric to measure the minimum level of anonymity that a session offers.

Choosing helper nodes . In fact, the source can determine which AS becomes the midway node by using AS-level network topology and AS-relationships available to the public [121], and obtain the anonymity set accordingly. Because the source node also possesses information about available middle nodes in the network, the source can improve the resulting anonymity-set size. The source first selects a subset of the helper nodes, computes the corresponding midway nodes and their respective anonymity-set sizes, and uses the helper node that yields the largest anonymity-set sizes.

However, it is dangerous for the source to use all or most of the available helper nodes to optimize the anonymity set, because an adversary located between the source and the midway node can reduce the anonymity set of the destination based on the selected helper nodes.

4.3.4 Integrating Payload Encryption

PHI explicitly requires payload encryption. To defend against session hijacking attacks, PHI binds a source's DH public key to an established path. Upon setting up each session, the source node randomly generates a pair of DH keys for negotiating shared symmetric keys with the midway node and the destination. As opposed to Dovetail, PHI uses the hash of public key as the session identifier to bind the encryption key to the session itself. As a result, an adversary that hijacks the connection by simply replaying headers with the adversary's payload will be detected immediately by the destination.

Not only the destination, but also the midway node needs to check that the session identifier is associated with the source node's public key. If the midway node fails to verify that the session identifier matches the public key encrypting the destination's address, a node between the source node and the midway node can still hijack the session setup process: the midway node can replace the session identifier with the hash of its own public key and circumvent the destination's verification.

Note that the midway node and the on-path nodes after the midway node can still launch hijacking attacks. During setup sessions, they can replace the session identifiers with hashes of their own public keys. However, these nodes have no incentives to launch such attacks because they already obtain full knowledge about the destinations, shown in Table 4.1.

4.4 PHI Protocol Details

In this section, we describe PHI's packet format and its session setup phase in detail. We also briefly state how PHI achieves sender-receiver anonymity.

4.4.1 Packet Format

There are 4 types of packets in PHI: path requests, midway requests, path replies, and forward/backward data packets. As a network-layer protocol, all PHI headers are placed after the respective layer 2 headers. A path request allows a node to create or extend a path towards its intended entity while concealing its network location. A source node uses path request to create the first half path to the helper node that it chooses. A midway node uses the path request to extend the path to reach the destination to create a full path. A midway request, as its name indicates, is used by the helper node to perform the back-off method described in Section 4.3.3 to find the midway node. When a destination receives a full path established from the source, the destination uses a path reply to send the path back to the source so that they can start transmitting actual data using data packets. Figure 4.5 graphs a reference packet header format in detail.

Figure 4.5 demonstrates a reference packet header formats for PHI headers, which we use in our prototype implementation.

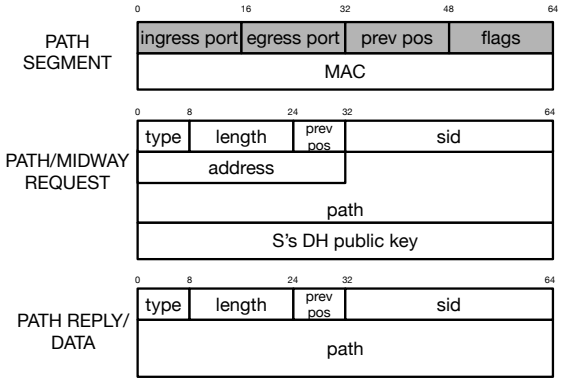


Figure 4.5: Formats of a path segment and packet headers. The shaded fields are encrypted using a node's local secret key.

Common header fields . In all PHI packet headers, there are 4 common fields: type, length, previous hop's position, and a session identifier. The type field specifies which type the PHI packet header belongs and whether it uses a normal path or a long path. The length field describes how long the header is. The previous-hop-position field records the position of the last-hop node's path segment so that the current node can verify its path segment. Lastly, the session identifier helps identify which session the packet header belongs to and should be randomly generated by a source node.

Path . Another basic building block of PHI headers shared by all types of PHI packets is a PHI path. A PHI path contains a sequence of path segments. As shown in Figure 4.5, a path segment is a 16-byte block containing an 8-byte encrypted information block and an 8-byte MAC. An encrypted information block contains an ingress port field, an egress port field, a previous-hop position field, and flags. The first two fields help a node keep records of where to forward a packet, and the previous-hop field assists a node in finding the path segment of the previous-hop node so that the node can verify its MAC. Lastly, the flags help a node to keep record of its role. For example, the midway node sets the "midway flag" so that it remembers its role when processing a data packet.

Path/midway requests . In addition to common headers, both path and midway request headers also have a 4-byte address field, a path, and the source node's DH public key. For a path request from a source to a helper node, the source node fills the address field with the helper node's address so that on-path nodes can decide how to forward the path request. The actual destination address is encrypted using

the shared key between the source node and the helper node, and appended to the path request. For a path request from a midway node to the destination, the address field contains the destination's address. The address field of a midway request also contains the destination address, which is used in the back-off technique for an on-path node to decide whether it is the midway node. With respect to the DH public key, both the helper node and the destination node use it to derive shared symmetric keys with the source node. We use ECDH key exchange protocol for small public key sizes.

Path replies/data packets . A path replies and a data packet share the same header format. The only field in addition to the common header is an end-to-end path. The selected destination and the midway node are already implicitly specified in a path. Therefore, there is no need to specify their addresses explicitly.

4.4.2 Managing Keys

Each node in the network maintains a master secret key, from which it can derive three keys: a position key k^{pos}, an encryption key k^{enc}, and a MAC key k^{mac}. The position key is used by the node to determine the position of its path segment in a packet header. The encryption key encrypts the routing information in a path segment. The MAC key computes a MAC that helps a node to verify its path segment.

The destination node needs to generate a pair of DH keys. To communicate anonymously with the destination, a source node needs to obtain the destination's public key through an out-of-band channel. In order for a source to communicate with a destination in a new session, the source also needs to generate a new pair of DH keys for the session to negotiate with the destination a shared symmetric key to encrypt data payloads.

4.4.3 PHI Session Setup

There are four steps in PHI's session setup phase. First, the source node selects a helper node and establishes a half path to the helper node. Second, the helper node uses the back-off technique to search for a midway node. Third, the midway node extends the existing path to become a full end-to-end path between the source and its destination. Finally, the destination sends the created path back to the source. In this section, we walk through details of all four steps in an Internet setting. We remark the same process can be adapted to other network architectures.

Constructing a path to the helper node To communicate with a destination node D, the source node S first selects a helper node M. If it is the first time for S to use M, S also needs to retrieve M's public key

with its associate certificate. For example, S can obtain the information about the helper nodes' public keys from directory services like those employed in Tor [79]. S needs to verify M's public key before using it.

S then creates a path request. S first randomly generates a pair of DH keys for the session and uses the hash of the public key as the session identifier. Then, S generates random bits to initiate the path. S also encrypts the destination address using M's public key and puts the resulting ciphertext as the packet's payload. At last, S fills the address field in the path request header with M's address and sends it towards M.

Depending on the required success rate, S repeats the above process to create a number of setup packets in case that a single setup packet cannot achieve the desired success rate. S sends all setup packets to M together and then waits for responses. In addition, because the setup process can fail, S keeps a timer so that it can resend setup packets.

When receiving a path request, each on-path node follows the procedures described in Section 4.3.2. Based on the address specified in the packet header, an on-path node first creates its path segment following Equation 4.4. Then the node determines its path segment's position in the header field by Equation 4.1 and replaces the current path-segment with its own. Afterwards, the node can proceed to forward the packet towards M. The same process repeats itself until the packet reaches M.

Back-off to find the midway node After M receives a path request from S, M first verifies that the session identifier is indeed the hash of the session's public key in the payload. If so, M decrypts D's address. M creates a midway request packet by placing D's address in the header's address field and using the path in the path request. M then sends the midway request back to the AS where the path request comes from.

An on-path node receiving the midway request first retrieves its segment inserted into the path. It decrypts the routing information and checks the corresponding MAC. If the MAC in the path segment successfully verifies, the node decides whether it is the midway node based on D's address, the previous-hop AS, and its local policy. If becoming a midway node violates its policy, it simply forwards the midway request toward S using the ingress port in the path segment.

Create a full path If a node decides to be the midway node, the node creates a new path segment. It changes the egress port field to the one towards D, and sets the $midway$ flag in the segment. Then, it replaces the current segment with its newly created one. Finally, it creates a new path request that addresses to D's and contains the path and the session's public key, and forwards it to D.

Nodes between the midway node and D process the path request in the same way as the nodes between S and M. They calculate the positions of their path segments, insert their new path segments accordingly, and forward the packet.

Path replies When D receives the path request sent by the midway node, D first verifies that the public key matches the session identifier. Then, D conducts a DH key exchange operation to generate a symmetric key shared with S. The key is used as the master key to encrypt payloads of subsequent data packets. D uses the path composed in the forward path to send a path reply back to S. Every on-path node, including the midway node, retrieves its path segment, obtains its routing information, and forwards the packet towards S. Finally, when the path reply reaches S, S can retrieve the path and construct data packets with it.

4.4.4 Sender-Receiver Anonymity

PHI achieves sender-receiver anonymity defined by Pfitzmann and Köhntopp [144] using an indirection node, called rendezvous node. In order for a destination node to conceal its identity, it first chooses an available rendezvous node and establishes a session with it. After obtaining the path between the rendezvous node and itself, the destination node can publish the path together with the address of the rendezous node through an out-of-band channel, like a website, so that a source node intending to connect to the destination node can retrieve the path.

For a source to connect to the anonymous destination, after fetching the path and the rendezvous node's address, the source first establishes a path between the rendezvous node and itself. Then the source can place both two paths in its data packets sent to the destination node. When the rendezvous node receives a data packet from the source, it forwards the packet following the path established by the destination.

4.5 Security Analysis

In this section, we first quantitatively analyze in detail the security of PHI compared to VSS used in LAP and Dovetail under topology-based attacks. Then, we discuss PHI's defense mechanisms against various known active and passive attacks.

4.5.1 Defending Against Topology-based Attacks

We compare the probability that an adversarial AS successfully discovers a packet's source node in PHI's segment-position randomization method and VSS. VSS differs from segment-position randomization in that it provides to the adversary additional information about the number blocks already inserted. To quantitatively compare between PHI and VSS's defenses against topology-based attacks, we conduct an experiment with the Internet topology considering the IPv4 address space.

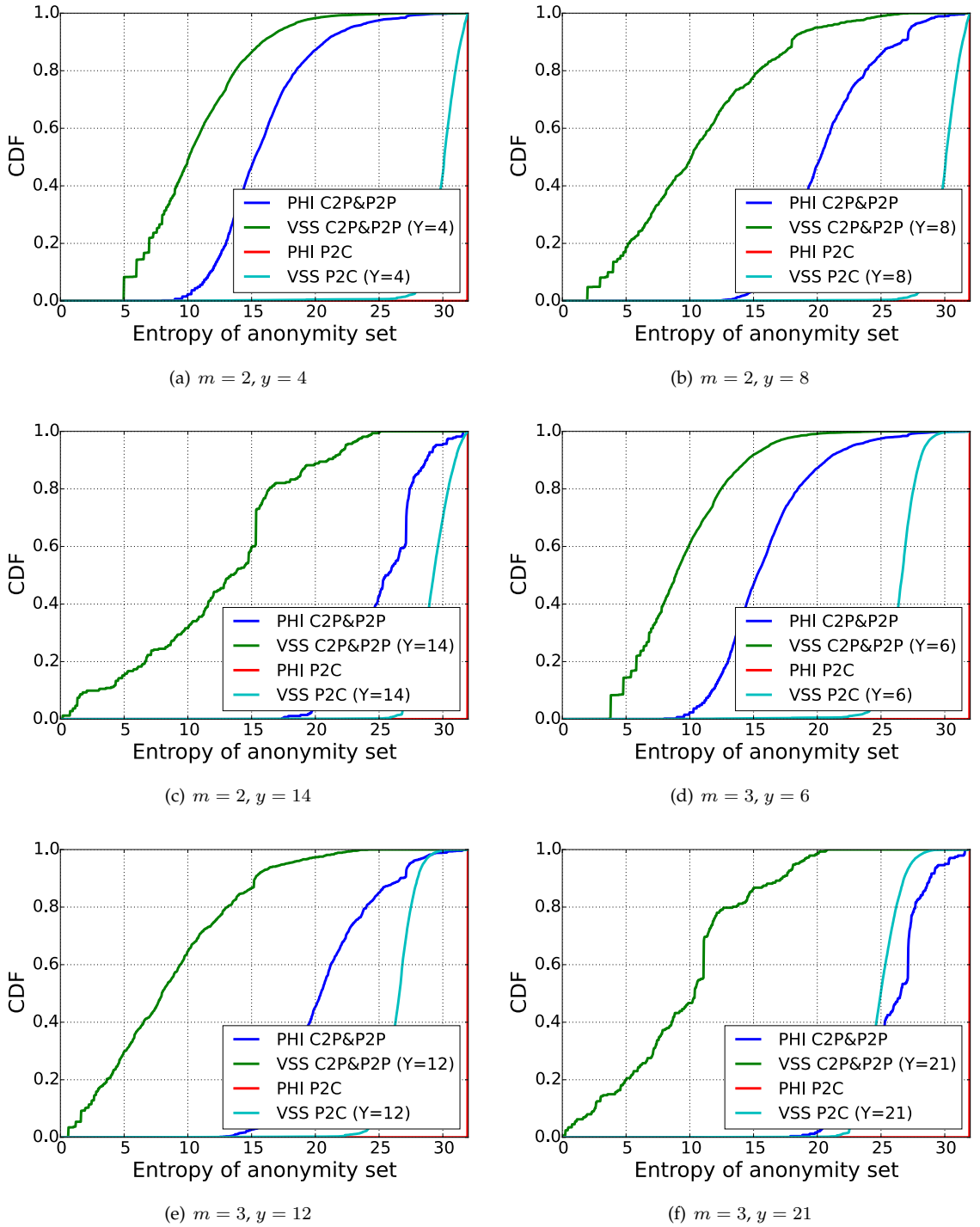
Modeling anonymity-set size for VSS technique We model the probability $P(x = S|Y = y, m = l)$ that an adversarial AS can locate a source node S given that the number of blocks in a header $Y = y$ and the maximal number of blocks that can be occupied by a path segment $m = l$. $P(D = d)$ is the distribution of distances between the source of adversarial ASes. $P(Y = y|D = d, m = l)$ is the conditional probability of d nodes output a sequence of path segments with total length y. $P(x = S|D = d)$ is the probability that the source can locate S if it is d hop away from S. d_{max} is the maximal diameter of the network. We use the following equation to compute $P(x = S|Y = y, m = l)$ from $P(D = d)$, $P(Y = y|D = d, m = l)$, and $P(x = S|D = d)$:

$$\begin{aligned} P(x = S|Y = y, m = l) &= \frac{P(x = S|m = l)}{P(Y = y|m = l)} \\ &= \frac{\sum_{d=1}^{d_{max}} P(x = S|D = d, m = l)P(D = d)}{\sum_{d=1}^{d_{max}} P(Y = y|D = d, m = l)P(D = d)} \\ &= \frac{\sum_{d=1}^{d_{max}} P(x = S|D = d)P(D = d)}{\sum_{d=1}^{d_{max}} P(Y = y|D = d, m = l)P(D = d)} \end{aligned} \tag{4.9}$$

We define the equivalent anonymity-set size observed by the adversarial AS as $\frac{1}{P(x=S|Y=y,m=l)}$.

Experiment setup We use the CAIDA AS-relationship dataset [2] and the RouteView dataset [18] to construct an AS-level Internet topology annotated with both AS relationship and each AS's address space sizes. In our experiment, for each AS α, we compute the anonymity set observed by a neighboring AS β receiving packets from α in one of three cases: 1) β is α's provider (C2P), 2) β is α's peer (P2P), and 3) β is α's customer (P2C). In our experiment, we assume that every IPv4 address has equal probability to send the packet. As a result, the anonymity-set sizes are calculated as the sizes of IPv4 address spaces. We also assume that ASes obey the valley-free policy when forwarding packets. We compare PHI and VSS by the cumulative distribution of the anonymity-set sizes yielded by both schemes.

To compute anonymity-set sizes observed by AS β when using PHI, we add the anonymity-set sizes of all ASes that can send packets traversing the link from α to β without breaking the valley-free policy. For example, in Figure 4.1, if AS2 is α, AS3 is β, and α is β's customer, the anonymity-set size observed by



(a) $m = 2, y = 4$ (b) $m = 2, y = 8$

(c) $m = 2, y = 14$ (d) $m = 3, y = 6$

(e) $m = 3, y = 12$ (f) $m = 3, y = 21$

Figure 4.6: In VSS, we vary the maximal number of path-segment blocks per hop, denoted as m, and the observed total number of path-segment blocks Y. We also distinguish between the case where the adversary received a packet from its customer (C2P) or peer (P2P) and the case where the adversary received a packet from its provider (P2C). We consider IPv4's address space, so the maximal entropy is 32. Note 99.9% of paths are less than 8 hops long according to Section 4.6.

β in this case is the sum of the IPv4-address-space sizes of all customers of AS2 including AS2 itself, i.e., {AS0, AS1, AS2, AS4, AS5}. For VSS, we compute the equivalent anonymity-set size observed by β when receiving a packet from α.

We can compute $P(x = S|D = d)$ directly from the topology by add up the anonymity-set sizes of the ASes that are d hops away from β and can send packets through link $\alpha \rightarrow \beta$ following the valley-free policy. For example, in Figure 4.1, if AS3 is AS2's provider and it is malicious, and $d = 3$, the anonymity-set size is simply the AS0's address-space size. $P(D = d)$ is inversely proportional to $P(x = S|D = d)$ because we assume that every IPv4 address has the same probability to originate the packet. Finally, if we additionally assume that each AS uniformly and independently picks the size of its path segment, we can derive $P(Y = y|D = d, m = l)$ by combinational algebra.

Results Figure 4.6 graphs the resulting CDF of observed anonymity-set sizes in the form of entropy. Each sub-figure draws the CDF of PHI and VSS with different m and Y. We also vary inter-AS relationship between malicious ASes and their neighbors. The results from C2P and P2P are exactly the same, therefore we use the same line to represent both of them in each sub-figure. For fairness, when calculating PHI's CDF in each sub-figure, we only consider β who can actually observe that $Y = y$ when $M = m$.

In general, PHI provides 30–60000 times larger anonymity sets than VSS when the relationship is C2P or P2P, and 4–60 times larger anonymity sets than VSS when the relationship is P2C. The differences generally enlarge with larger values of m and y. When $m = 2$ and $y = 4$, PHI's anonymity-set size is around 30 times larger than VSS's for C2P or P2P, and 4 times larger for P2C. When y increases, the gap grows wider. In fact, when $m = 2$ and $y = 14$, PHI offers a 250 times larger anonymity-set size than VSS for C2P or P2P, and 5-6 times larger for P2C. The gap further increases with a larger m. When $m = 3$ and $y = 21$, PHI provides an anonymity set that is almost 60000 times larger than those for C2P & P2P, and 70 times larger for P2C.

We also observe that PHI provides an anonymity-set size of almost 2^{32} when the AS relationship is P2C. This makes sense in a well connected network like the Internet. For example, in Figure 4.1, if AS2 is AS3's provider and AS3 is compromised, every packet that AS2 sends to AS3 can potentially come from the entire Internet with the valley-free policy preserved. The observation implies that choosing the destination node's providers or topological ancestors can be as secure as selecting an arbitrary third party end host without breaking the valley-freeness rule and introducing a longer path.

Impact of inaccurate simulation As described in Section 4.3.3.2, the anonymity-set size depends on the anonymity-set size of the source node observed by the midway nodes. Thus, an inaccurate topology

simulation might direct the source node to pick a wrong helper node, and thus a wrong midway node, which can potentially reduce the anonymity-set size.

4.5.2 Attacks Against Anonymity

With respect to the case where an adversary only controls a single AS, PHI can resist various passive or active attacks against anonymity besides the topology-based attacks discussed in Section 4.5.1. By passive attacks, we refer to the attacks that only require observation or logging of the traversing packets. In comparison, an adversary that conducts active actions can delay, drop, or modify traversing packets, to impact the anonymous traffic.

PHI defends against the following passive attacks in addition to topology-based attacks:

- **Observing packet payloads**. Packet payloads contain various identifiers, such as browser cookies, which can help correlate packet flows and deanonymize users. PHI defeats such attacks by requiring the source node to encrypt end-to-end all the traffic between itself and the destination node. In addition, all data packets are padded to a fixed size to prevent information leakage in PHI.

- **Session linkage**. If an adversary correlates sessions initiated by the same source node, the adversary can profile the user and subsequently deanonymize him or her. PHI prevents this attack by requiring the source to generate a random session identifier for each session. Hence, even if multiple sessions from the same source node follow the same path, the resulting path segments differ. Moreover, the source also generates a new pair of keys for end-to-end encryption to prevent session linkage from a set of colluding destinations.

PHI also defeats the following active attacks:

- **Session hijacking attacks**. A compromised on-path node can hijack a session by replacing the session's public key with its own. PHI derives the payload encryption key from the session's public key, which in turn is bound to the destination address. For nodes between a source node and a helper node, to whom the destination address is unknown, replacing the session's public key is detected by the helper node when decrypting the destination's address is unsuccessful.

- **Packet-header modification**. Adversarial nodes can modify packet headers to change the paths that the packets traverse. If the adversary directly modifies the path embedded in the packets, benign nodes can detect the changes because the verification of the MAC in its path segment fails. During the session setup, an adversary can also attempt to modify the address field in a path request or a

midway request. Nevertheless, without colluding ASes, modifying the address field only disrupts the session setups, but does not reduce the observed anonymity set.

- **Replay packets**. An adversary node can replay packets. However, the replayed packets only traverse the same path that the previous packets traverse, yielding no new information about the either end of the session.

- **Pre-computation attacks**. An adversary can send many packets with different session identifiers to determine the positions of AS segments. PHI defeats the pre-computation attack by requiring that sources generate fresh session identifiers and ASes update their master keys periodically.

4.6 Evaluation

In this section, we first simulate and compare the overhead of PHI and VSS with Internet data traces from iPlane [12] and CAIDA [1]. Then we evaluate the performance of HORNET, LAP, Dovetail, and PHI on a high-speed software router.

4.6.1 Comparison of Overhead between PHI and LAP with VSS

We simulate the bandwidth overhead of both PHI and VSS-based LAP using iPlane traceroute data [12] and CAIDA anonymized Internet traces obtained by monitoring on high-speed backbone links [1]. Our CAIDA data traces include all traffic from a backbone link between 13:00 to 14:00 on Mar. 20th, 2014.

Path lengths We first analyze the distribution of the number of setup packets required by PHI. We obtain the distribution of total number of hops between source and destination by using iPlane traceroute data. For each trace, we count the number of AS hops. Our results show that the mean AS-path length is 4.2, and 99.99% of paths are within 7 AS hops. Thus, we choose 7 as the maximum length of a normal AS path, and paths with lengths larger than 7 are long paths. With regard to paths longer than 7, setting $m = 48$ can achieves 90% success rate when 5 setup packets are used.

Packet-header sizes We use CAIDA anonymized Internet traces to compute the probability that a setup phase succeeds given different packet-header sizes. We vary the maximum number of path segments m and apply Equation 4.7 to each flow of AS path length r. Our result show that when 4 setup packets are used, a setup phase succeeds with probability 75% when $m = 8$. The probability of success increases to 90% when $m = 12$.

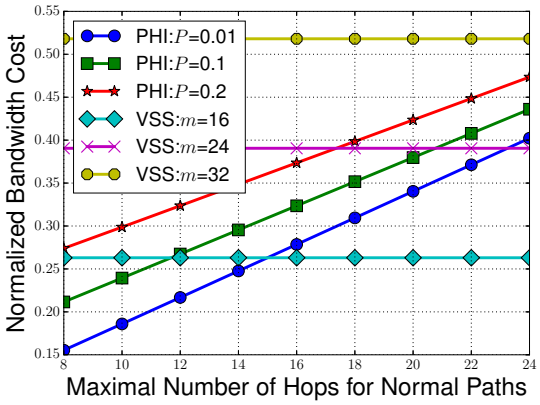


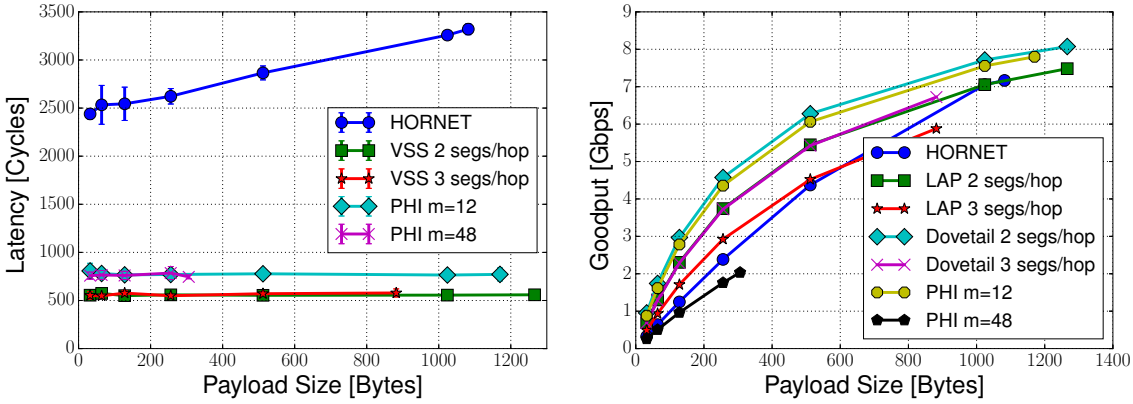
Figure 4.7: Bandwidth cost of PHI under different configurations. Lower is better. For PHI, we vary the number of path segments m that a header can contain for each line. Different lines have different probability P to select a large header for a short path. We fix the size of large headers so that they can contain 24 path segments. For comparison, we also draw VSS's bandwidth cost with different header sizes. All the bandwidth cost is normalized by dividing the bandwidth cost without anonymity.

Bandwidth Overhead We then evaluate the bandwidth overhead caused by larger headers in VSS compared to the overhead of additional setup requests in PHI. We conduct a trace-based simulation of both schemes using our CAIDA dataset. We assume all the flows in the dataset are converted to PHI and VSS sessions and we compute the extra bandwidth overhead introduced by both schemes with different configurations. Figure 4.7 demonstrates the resulting bandwidth cost. For PHI, we vary two parameters: the number of path segments that a normal header contains m and the probability that a larger header $m = 48$ is used for a short path P. We also test the bandwidth cost of VSS with different packet header sizes corresponding to an amplification factor of 2 to 4.

We observe that the bandwidth overhead increases as m and P get larger. With small headers where $m = 8$ and $P = 0.01$, PHI's bandwidth cost is 43% smaller than VSS with an amplification factor $A = 2$. When end hosts increase P to offer stronger security for sessions with longer paths, the bandwidth cost is still small. When $m = 8$ and $P = 0.2$, the bandwidth cost of PHI is only as large as that of VSS with $A = 2$ and at least 65% smaller than VSS with larger A. When latency of setup phases is concerned and $m = 12$ is used, the bandwidth cost of PHI is 11% smaller than VSS with $A = 2$ when $P = 0.01$. In the following evaluation, we choose $m = 12$ for a normal header and $m = 48$ for a large header.

4.6.2 Performance Evaluation

Implementation and testbed setup We implement PHI on a high-speed software router using Data-Plane Development Kit (DPDK) (version 2.1.0) [8]. For comparison, we also implement HORNET, LAP, and Dovetail in the same setup.



(a) Latency (b) Goodput

Figure 4.8: a) **Latency of processing data packets in HORNET, VSS, and PHI when different payload sizes vary**. Lower is better. b) **Goodput of HORNET, LAP, Dovetail, and PHI with different payload sizes on a 10 Gbps link**. Higher is better. For VSS, we test the configurations of 2 path segments per hop and 3 path segments per hop. For PHI, we evaluate the cases where m equals 12 and 48 respectively.

To accelerate the underlying cryptography, we use Intel AESNI technology. In our implementation, we use 128-bit AES in CBC mode and 128-bit AES CBCMAC as our respective encryption and MAC functions. To implement the DH key exchange protocol and public key cryptography in HORNET, we use curve25519-donna library [7].

We test our implementation on a testbed composed of a high-speed software router and a Spirent TestCenter traffic generator/monitor [20], which are connected by twelve 10 Gbps links. Our software router has Intel Xeon E5-2680 CPU with 2 sockets and 8 cores per socket, 64 GB RAM, and 3 Intel 82599ES NICs supporting in total 120 Gbps maximal throughput. In all of our experiments, we configure our DPDK library to assign 1 CPU core to each port to process incoming packets.

Setup latency We first measure the latency of processing setup requests in HORNET and in PHI. Table 4.2 shows the latency for processing setup requests for HORNET and PHI with different packet sizes. We observe that processing a HORNET session request requires almost 800 times more time than processing a PHI because the former needs a DH key operation but the latter only needs symmetric cryptography. Moreover, processing setup requests of different sizes incur similar latency, because the amount of computation needed is independent from header sizes. A single core can process PHI setup requests/replies at a speed of up to 3.76 Mpkt/s on a single core. According to our analysis of the CAIDA data trace used in Section 4.6.1, the average number of flow initiation on a 40 Gbps backbone link is 1721 per second. Hence, PHI can potentially satisfy the requirement of future traffic at today's budget.

Scheme	Cycles	μs
HORNET	$(6.60 \pm 0.28) \times 10^5$	$(2.00 \pm 0.09) \times 10^2$
PHI (m = 12)	821 ± 83	0.249 ± 0.025
PHI (m = 48)	962 ± 132	0.291 ± 0.040

Table 4.2: Latency for processing setup packets in HORNET and PHI.

Data-forwarding latency We evaluate the latency for forwarding data packets with various payload sizes by a single core on a 10 Gbps link using PHI, VSS, and HORNET. We base our comparison on LAP using VSS to hide path information. The latency of Dovetail data forwarding should yield similar results because both LAP and Dovetail require decrypting a single block and check a MAC.

As shown in Figure 4.8(a), in general, the latency needed by processing a HORNET data packet is 3 times higher than that of PHI. The latency increases linearly when the payload size grows, because HORNET data forwarding process encrypts or decrypts the entire header and payload. With respect to the comparison between PHI and VSS, PHI consumes on average 30% more time to forward a data packet than VSS as PHI requires additional PRF computation to locate the current path segment within the header. However, differing from HORNET, the computation required by PHI and VSS is constant and does not increase together with enlarging payloads.

Goodput. We also measure the goodput of different schemes on a 10 Gbps link with respect to various payload sizes. Figure 4.8(b) graphs the results. There are two factors impacting the goodput of a protocol: forwarding speed and header sizes. Higher forwarding speed yields higher goodput while smaller header sizes help increase goodput. Yet, there is a third factor in our experiment that limits the goodput of LAP, Dovetail, and PHI: the maximal transmission rate on the link. In fact, LAP, Dovetail, and PHI can achieve 10 Gbps even with the payload size as small as 32. Hence, only the header size will influence their goodput.

We observe that PHI with $m = 12$ and Dovetail using 2 path segments per hop achieves the highest goodput among all the evaluated schemes because their efficient data forwarding and small headers. Compared to LAP with 2 path segments per hop, they achieve 18% and 12% more goodput. As larger headers are used, all three schemes' goodput decreases. On the other hand, because HORNET's throughput is lower than the maximal throughput, its goodput increases faster as the payload sizes increase. In general, HORNET's goodput is 60% lower than PHI with $m = 12$.

Maximum throughput Lastly, we evaluate the maximum throughput on all 12 ports, which can possibly switch packets at the speed of 120 Gbps. We observe that LAP, Dovetail can fully saturate the 120 Gbps even with the smallest packets possible for individual protocols. In comparison, HORNET can only

provide at maximum 90 Gbps with 1500-byte packets. But this number drops to around 50 Gbps when the packets contain payloads of 32 bytes. PHI's maximum throughput is very close to LAP and Dovetail. PHI can offer 106 Gbps with 32-byte payloads, and it can saturate 120 Gbps when using payloads larger than 128 bytes. In summary, PHI data forwarding can achieve similar maximum throughput as those of LAP and Dovetail, and can offer 35–200% speed-up against HORNET depending on packet sizes.

Limitations The performance results from our single-node testbed indeed apply to a larger testbed, because the amount of computation required by forwarding is independent of traffic types or a node's position in the network. However, there are several limitations of our single-node evaluation compared to an end-to-end evaluation on a larger testbed. First, given the small computation latency of PHI, propagation latency will dominate end-to-end latency. An end-to-end evaluation will better demonstrate user-experienced latency. Second, the single-node evaluation cannot reflect real networks' heterogeneity such as connectivity and throughput.

Chapter 5

TARANET: Traffic-Analysis Resistant Anonymity Protocol at the Network Layer

In this chapter, we propose TARANET, a scalable, high-speed, and traffic-analysis-resistant anonymous communication protocol, which uses the end-to-end traffic shaping assisted by packet splitting as one of its novel mechanisms. TARANET is directly built into the network infrastructure to achieve short paths and high throughput. It uses mixing for its setup phase and end-to-end traffic shaping for its data transmission phase to resist traffic analysis. In this chapter, we make the following contributions:

1. We propose an efficient end-to-end traffic shaping technique that maintains per-flow constant-rate transmission on all links and defeats traffic analysis attacks. We also propose in-network packet splitting as the enabling mechanism for the end-to-end traffic shaping technique.
2. We present an onion routing protocol that enables payload integrity protection, replay detection, and splittable packets, which are essential for end-to-end traffic shaping.
3. We design, implement, and evaluate the security and performance of TARANET. Our prototype running on commodity hardware can forward over 50 Gbps of anonymous traffic, showing the feasibility to deploy TARANET on high-speed links.

The material presented here is closely related to the paper published at EuroS&P'18 [58].

5.1 Problem Definition

We consider a scenario where a government secretly conducts a network mass-surveillance program. By stealthily tapping into inter-continental fiber links, or by controlling a set of domestic ISPs/IXPs, the government gains bulk access to network traffic. Besides matching identifiers to filter packets, the government is also capable of conducting traffic manipulation and traffic pattern matching. A pair of

anonymity-conscious users would like to communicate through the network, hiding the fact that they are communicating with each other. The communication between the pair of users is bi-directional. Without loss of generality, we call the user that initiates the anonymous communication *sender*, and the other user *receiver*.

5.1.1 Network Assumptions

The underlying network is divided into ASes, or simply *nodes*. Each node forwards a packet according to a routing segment. Each routing segment contains forwarding information for a node between the sender and the receiver. For a sender to reach a receiver, the sender can obtain a sequence of *routing segments*, named *path*.

Except the ingress and egress links that are necessary to a forwarding node, routing segments should leak no extra information about the end hosts or the path taken to the forwarding node. This property is satisfied by several next-generation Internet architectures that use source-controlled routing (e.g., NIRA [195], Pathlet routing [92], or SCION [198]), or in today's Internet through IPv6 Segment Routing [19].

5.1.2 Threat Model

We consider a global active adversary. We assume an adversary capable of controlling all links between any pair of ASes, or between an AS and end hosts. This means that the adversary has bulk access to contents and timing information of packets on all links and can also inject, drop, delay, replay, and modify packets. We additionally assume that the adversary is able to compromise a fraction of ASes. By compromising an AS, the adversary learns all keys and settings, has access to all traffic that traverses the compromised AS, and is able to control the AS including delaying, redirecting, dropping traffic, and fabricating, replaying, and modifying packets. However, we only guarantee relationship anonymity for end hosts if there exists at least one uncompromised AS on the path between sender and receiver. We remark that the adversary under this assumption is able to perform all traffic analysis attacks in Section 2.3.

5.1.3 TARANET Goals

Anonymity TARANET aims to provide relationship anonymity (defined by Pfizmann and Köhntopp [144]) when a sender and a receiver share mutual trust. We refer to the relationship anonymity under this condition as *third-party relationship anonymity*. While requiring trust in receivers limits our protocol's application

scope, third-party anonymity is actually sufficient when communicating parties are authenticated end-to-end (e.g., VoIP), when avoiding censorship where the receiver (e.g., a foreign news site) is known not to cooperate with the censoring entity, when a warrant canary (e.g., `pushover.net/canary.asc`) has been recently updated for that endpoint, or when the receiver is a trusted node acting as a proxy.

High throughput and low latency The latency incurred by processing packets should be small enough not to interfere with interactive low-latency traffic. Additionally, TARANET's throughput should be on par with that of today's core routers.

Scalability Nodes should be capable of handling the large volume of simultaneous connections as observed on Internet core routers. TARANET aims to minimize the amount of per-flow state maintained. Specifically, TARANET guarantees that the amount of state on a router is bounded given a fixed throughput. Moreover, adding new nodes to the network should additionally not require coordination with all other nodes.

5.2 Protocol Design

Communication Model Hosts communicate anonymously through TARANET-enabled Autonomous Systems (ASes) using *flowlets*. A TARANET flowlet allows an end host to send traffic anonymously at a constant rate B for a fixed time period T. All anonymous traffic is divided into a set of flowlets by end hosts to leverage TARANET's service. Figure 5.1 graphs the lifecycle of a TARANET flowlet.

A flowlet's life-cycle begins with a *setup phase* followed by a *data transmission phase*. At the beginning of the setup phase, a sender first anonymously retrieves two paths: a forward path from the sender to the receiver and a backward path from the receiver back to the sender. A path contains the routing segments, the public keys, and the certificates of all nodes between the two end hosts. One mechanism for anonymously retrieving paths is to have end hosts query global topology servers through TARANET flowlets that are established using network configuration information (e.g., distributed to end hosts through a DHCP-like infrastructure [57]). Another mechanism is to disseminate paths and public keys throughout the network to end hosts, as done in certain future network architectures (e.g., NIRA [195], Pathlets [92]). A third mechanism could be based on private information retrieval (PIR) [61], which allows to trade off a lower communication overhead for an increased computation overhead on the servers providing the network information and the keys.

Once the sender successfully obtains both paths, the sender and the receiver exchange two setup messages traversing the obtained paths. By processing a setup message, each on-path node establishes a

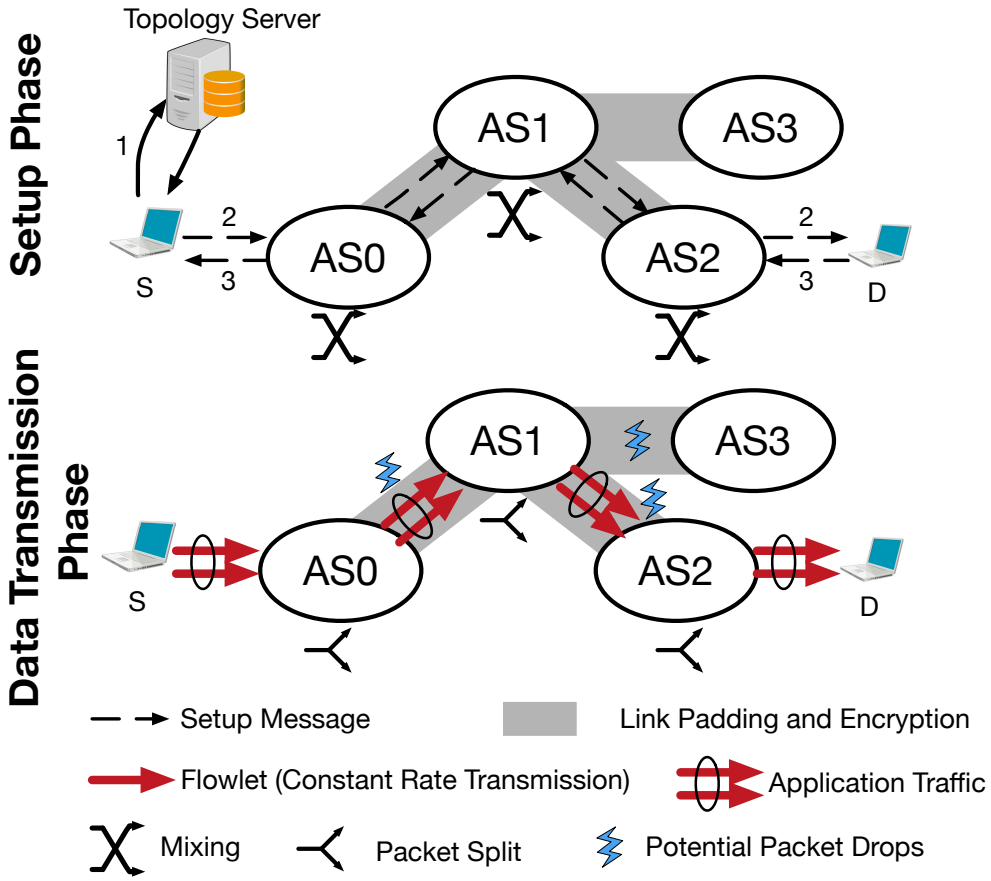


Figure 5.1: TARANET design overview.

shared symmetric key with the sender. The per-node shared key is later used to conceal routing information by layered encryption/decryption in the data transmission phase. To prevent storing per-flow cryptographic state on each node, a node encrypts the shared key using a secret key that the node never reveals. The resulting encrypted shared key, which we call the *Forwarding Segment* (FS), is carried by all data packets and allows the node to dynamically retrieve its shared symmetric key.

With routing segments, FSes, and per-node symmetric keys, the sender is able to create TARANET data packets that can reach the receiver. An on-path node can process a data packet with only symmetric cryptographic operations, enabling highly efficient packet forwarding. Within the first batch of packets along the forward path, the sender transmits all routing segments, FSes, and shared symmetric keys for the backward path, so that the receiver becomes capable of sending data packets backwards.

Traffic analysis resistance TARANET resists traffic analysis attacks by combining an onion routing protocol (an enhanced adaptation of the one in HORNET), a newly proposed end-to-end traffic shaping scheme, and mixing. First, compared to HORNET which provisions confidentiality, authenticity, and

bit pattern unlinkability, TARANET additionally offers payload integrity protection, replay protection, and packet splitting, which is a vital enabling technique for the end-to-end traffic shaping scheme (Section 5.2.1). Second, for the data transmission phase, TARANET enables end-to-end traffic normalization for flowlet traffic. For each flowlet, the sender and receiver maintain a constant transmission rate shared by every end host. Each forwarding node maintains the same constant transmission rate for outgoing packets belonging to the flowlet (Section 5.2.2). Third, for messages in the setup phase, TARANET requires each node to conduct mixing [56] in order to prevent linking messages based on their timing and order (Section 5.2.3). Finally, to hide the difference between setup packets and data packets and to defeat a global eavesdropper that monitors number of flowlets on links between nodes, TARANET additionally requires neighboring nodes to perform link encryption and link padding (Section 5.2.4).

Our decision to adopt different techniques for the setup phase and the data transmission phase is based on our observation of the different performance requirements in these two phases. Regarding the setup phase, because of the large number of simultaneous connection setups, batching setup messages on a node only incurs a small delay for the setup phase. Moreover, because changing the order of messages received by a node has no impact on the performance of the setup phase, we can randomize the order of messages within each batch. Finally, because processing a setup message requires public-key cryptographic operations, adding excessive chaff traffic to hide the timing of two setup messages is computationally inefficient.

For the data transmission phase, on the other hand, because packet order is important for TCP performance, randomizing the message order severely impacts application performance. Additionally, because data packet processing is highly efficient, we can actively conduct traffic shaping on both end hosts and intermediate nodes by using chaff packets (Section 5.2.2).

5.2.1 TARANET Onion Routing Protocol

Like the HORNET onion routing protocol [57], the TARANET protocol offers bit-pattern unlinkability, payload confidentiality, and per-hop authenticity. Bit-pattern unlinkability eliminates any identifiers that facilitate packet matching. Payload confidentiality prevents leaking upper-layer sensitive information. Finally, each TARANET header contains per-hop MACs that protect the integrity of both the header and the payload, unlike HORNET, whose per-hop integrity guarantees only cover the header. Therefore, in TARANET, tampered or forged packets will be detected by benign nodes on the path and dropped immediately.

TARANET also adopts the scalable design of HORNET, i.e., using packet-carried forwarding state.

Storing per-flowlet state at core routers requires a large amount of high-speed memory, precluding scalability. Thus, in line with state-of-the-art network-layer anonymity protocols [57, 105, 156], TARANET embeds all necessary forwarding state (e.g., onion decryption keys, next-hop information, control flags) in packet headers instead of storing the state on routers.

Protocol	Bit-pattern unlinkability	Scalability	Payload Integrity	Replay Protection	Packet Splitting
HORNET	Yes	Yes	No	No	No
TARANET	Yes	Yes	Yes	Yes	Yes

Table 5.1: Comparison between TARANET and HORNET onion routing protocols

We highlight three new features that TARANET introduces for the data transmission phase compared to HORNET. First, integrity protection is extended to data packets' payloads, eliminating tagging attacks targeting at manipulating data payloads to create recognizable patterns. Second, data packets within the same flowlet have unique identifiers bound to the packets themselves, enabling replay protection. Third, TARANET allows an end host to create special chaff packets, each of which splits into two packets at a specific node. To all other nodes, the original packet and the resulting packets are indistinguishable from ordinary data packets in the same flowlet. Split packets should traverse the same path as other packets in the flowlet and their per-hop MACs should be correct at each downstream node. Splitting a chaff packet into multiple packets plays a vital role in the end-to-end traffic shaping technique described in Section 5.2.2. We defer the detailed description of the technical aspects of packet splitting to Section 5.3.

Replay protection In TARANET, each TARANET packet header is uniquely identifiable, enabling intermediate nodes to detect replay attacks by checking the header's freshness. Specifically, an intermediate node can retrieve 3 fields from each packet: (1) a shared secret with the sender, (2) a per-packet Initial Vector (IV), and (3) a per-packet expiration time. The first two fields together uniquely identify a packet and are used as input to membership queries and for the insertions to the replay detector. The third field is used to check and drop expired packets.

TARANET nodes detect replayed packets by maintaining a rotating Bloom filters composed of 3 subject Bloom filters, as described by Lee et al. [117]. A packet received at $t = [i \cdot \frac{TTL}{2}, (i+1) \cdot \frac{TTL}{2}]$ is checked against all 3 filters and is only inserted into i'-th Bloom filter, where $i' \cong i \mod 3$. The i-th subject filter is cleared at time $(3N + i) \cdot \frac{TTL}{2}$ (N is an integer). The rotating Bloom filter guarantees that each packet inserted has a lifetime between TTL and $\frac{3}{2}TTL$, where TTL is the maximum lifetime of a packet. To reduce cache misses and increase performance, we also use *blocked Bloom filters* [149] instead of standard Bloom filters.

Replay detection state is not per-flow state, since the size of the detector grows linearly with its node's bandwidth, and not with the number of flowlets traversing that node. The size of our detector is ∼15 MB[1] for a 10 Gbps link when the false positive rate is at most 10^{-6} and $TTL = 6$ s (the maximum packet lifetime we consider in Section 5.3.4.1). Each false positive result causes the corresponding packet to be dropped. Given that the packet drop rate of the Internet is around 0.2% [171], we could reduce the detector's size by allowing higher false positive rate.

5.2.2 End-to-end Traffic Shaping

Flowlet Our basic idea for defending data transmission against traffic analysis is to shape traffic from heterogeneous applications into constant-rate transmission. A flowlet is the basic unit through which an end host is able to transmit packets at a constant throughput B and for a maximum lifetime T. During the lifetime T of a flowlet, the end host always transfers packets at rate B, inserting chaff packets if necessary. More generally, if an end host needs to transfer data at rate B' for time T', it initiates a sequence of $\lceil T'/T \rceil$ flowlet batches, each of which contains $\lceil B'/B \rceil$ simultaneous flowlets.

An end host shuts down a flowlet before the flowlet expires when there is no more data to send. When shutting down multiple simultaneous flowlets, an end host pads each flowlet with a random number of packets to prevent linking the flowlets by their expiration times. A node erases local state and terminates a flowlet when there are no more packets in its outgoing packet queue.

The key property of a flowlet is to maintain constant transmission rates not only at end hosts but also on all traversing links, to achieve which the flowlet relies on end-to-end padding instead of link padding. While link padding, which enables a pair of neighboring intermediate nodes to inject chaff to maintain a constant sending rate on a link, is effective against a network adversary, it is insufficient in the case of compromised nodes, since they can distinguish chaff inserted by neighbors from actual data packets. To defend against compromised nodes, we need chaff packets that are indistinguishable from data packets. Because TARANET uses onion encryption as a basic building block, one can create such indistinguishable chaff only when possessing shared keys with all traversing nodes. Thus, only sending end hosts are able to create such chaff.

Necessity of packet splitting To achieve constant-rate transmission, every flowlet should ideally arrive and leave with rate B at every node. However, drops/jitter may cause the incoming rate to vary: a higher rate is absorbed by the queues, but a lower rate requires that the node be able to produce "extra packets", which need to resemble legitimate packets to any downstream node. This implies that these packets

[1]Computed using the CAIDA dataset described in Section 5.5.

must also be generated by the sender like end-to-end chaff. But since the sender cannot send at a rate higher than B, it cannot send additional packets for the nodes to cache and use when needed. The only option then seems to be to have very long queues, and let each node fill a significant fraction of them with packets when the transmission of the flowlet first begins, before the node starts forwarding packets for that flowlet. However, this requires far too much state, and also adds significant latency in terms of time to the first byte, which makes this option unfeasible. This apparent dilemma can be solved with a technique we call *packet splitting*.

The packet splitting technique allows an end host to create a packet that can be split into two packets at a specific intermediate node.[2] The resulting packets should be indistinguishable from other non-splittable packets. This requirement indicates that the resulting packets should still traverse the same path and reach the other end host. We present the algorithm to split packets in Section 5.3.

Traffic shaping for flowlet outgoing rate To enable end-to-end traffic shaping, for each on-path node n_i, an end host selects a slot in its transmission buffer with probability Pr_i^{split} and fills in a newly generated splittable chaff packet that will split at node n_i. As an optimization, the end host can also select a slot that already contains chaff packets and replace it with splittable chaff packets. When a node receives a packet that should be split at the node, the node performs the split and caches resulting packets in its chaff packet queue.

Each node maintains a per-flowlet chaff queue of cached chaff packets. To guarantee an invariant outgoing flowlet rate, nodes periodically output a data packet from the data packet queue. In case that the data packet queue is empty, the node outputs a chaff packet from the flowlet's chaff queue. We limit the chaff queue size by a maximal length L_{chf}. In the (unlikely) scenario where the chaff queue is also empty, a local per-flowlet failure counter h is increased. When h exceeds a threshold H negotiated during flowlet setup, the node terminates the flowlet. H is a security parameter of the flowlet that determines how sensitive the flowlet is against potential malicious packet drops.

When a node shuts down a flowlet, an intermediate node no longer receives packets from upstream nodes. It will first drain its local chaff packet queue and then terminate the flowlet when the threshold H is reached. We remark that such a termination process results in successive termination on nodes and small variable intervals between termination times on different nodes because of the variable number of cached chaff packets.

We remark that both the chaff queues and failure counters constitute per-flow state. Nevertheless, the

[2]The general packet splitting technique supports a n-way split. We consider only two-way packet splits because of limited Maximum Transmission Units (MTU) in the network.

amount of state stored on a node is bounded by the node's bandwidth. Because each flowlet consumes a fixed amount of bandwidth, a node with fixed total bandwidth is only capable of serving a fixed number of flowlets. Thus, the amount of state that a node maintains for its flowlets is bounded given its total available bandwidth. Accordingly, a node will have to refuse setup messages for new flowlets if its bandwidth is already fully occupied. We will evaluate the amount of state the queues require in detail in Section 5.5.

5.2.3 Mixing in the Setup Phase

Each TARANET node applies a basic form of mixing when processing setup messages. After a setup message is processed by an intermediate node, the node queues the message locally into batches of size m. Once there are enough setup messages to form a batch, the node first randomizes the message order within each batch and then sends out the batch.

Through batching and order randomization, a TARANET node aims to obscure the timing and order for setup messages. An adversary that observes both input and output setup messages of a non-compromised node cannot match an output packet to its corresponding input packets within the batch.

The batching technique introduces additional latency because the setup messages have to wait until enough messages are accumulated. Assume that r_{setup} is the number of incoming setup messages every second, the added latency can be computed as $\frac{m}{r_{setup}}$. Given the large number of simultaneous connections within the network, the introduced latency is very low, as shown by our evaluation in Section 5.5.2.

5.2.4 Link Encryption and Padding

Each pair of neighboring TARANET nodes agree upon a constant transmission rate upon link setup. The negotiated transmission rate determines the maximum total rate for data packets. When the actual transmission rate exceeds the negotiated rate on a link, the sending node drops the excessive packets. When the actual transmission rate is lower than the negotiated rate, the sending node will add chaff traffic. The chaff traffic inserted by an intermediate node to shape traffic on a link only traverses the link and is dropped by the neighboring node.

To prevent an adversary observing a link between two honest nodes from distinguishing chaff traffic from actual data traffic, all pairs of neighboring nodes negotiate a symmetric key through the Diffie-Hellman protocol, and use it to encrypt all packets transmitted on their shared link. This also makes setup messages and data packets indistinguishable.

As an optimization to reduce chaff traffic and improve bandwidth usage, we additionally allow neighboring nodes to agree on a schedule of transmission rates as long as transmission rate is detached from the dynamics of individual traffic rates. For example, because the actual link rate on a link often demonstrates similarity at the same time of different days, we can reduce the amount of chaff traffic by setting the transmission rate between $[t, t']$ to $\mathcal{B}_{[t,t']} + k \cdot \Sigma_{[t,t']}$. $\mathcal{B}_{[t,t']}$ is the historic average transmission rate between $[t, t']$, $\Sigma_{[t,t']}$ is the standard deviation for the transmission rate, k is a factor that allows administrators to account for temporal changes of the bandwidth usage.

5.3 Protocol Details

This section presents the details of TARANET data packet formats and processing functions. We show how to create a fixed-size packet that can be split into two new packets of the same size whose per-hop MAC can still be verified. Using the packet processing functions, we present the TARANET data transmission phase on end hosts and intermediate nodes.

5.3.1 Notation

We first describe our notation. In general, sym^{dir} stands for the symbol sym of a specific direction $dir \in \{f, b\}$, which is either forward (src to dst) or backward (dst to src). sym_i^{dir} indicates the symbol sym belongs to i-th node n_i^{dir} on the path in direction dir. For simplicity, we denote the set of all sym for a path p^{dir} as $\{sym_i^{dir}\}$. We also define a series of string operations: 0^z is a string of zeros with length z; $|\sigma|$ is the length of the string σ; $\sigma_{[m..n]}$ refers to the substring between m-th bit to n-th bit of string σ where m starts from 0; $\sigma_1 \parallel \sigma_2$ stands for concatenation of string σ_1 and σ_2. Table 5.2 summarizes the notation used in this chapter.

5.3.2 Initialization & Setup Phase

In the setup phase, the sender node aims to anonymously establish a set of shared keys $\{s_i^{dir}\}$ with all nodes on the forward and backward path, and a shared key s_{SD} with the receiver. In the following protocol description and in our implementation, we use HORNET's Sphinx-based single-round-trip setup [57]. Note that we can also set up flowlets using Tor's telescopic method [79] which increases latency, but preserves perfect forward secrecy.

Once the setup phase is complete, in addition to the shared keys, the sender also obtains from each node on both paths a *Forwarding Segment* (FS) [57, Section 4]. The FS created by the node n_i^{dir} contains the key shared between the sender and that node s_i^{dir} and the routing information R_i^{dir} which tells the node

Symbol	Meaning
k	security parameter used in the protocol
c	size of per-hop segment
b	size of control bits and expiration time
r	maximum path length permitted by the protocol
m	fixed-size of a data packet payload
p^{dir}	path of a specific direction dir
l^{dir}	length of a path p^{dir}
n_i^{dir}	the i-th node on path p^{dir}
x_i^{dir}, $g^{x_i^{dir}}$	the private and public key pair of node n_i^{dir}
ENC	encryption function of a stream cipher
DEC	decryption function of a stream cipher
MAC	message authentication code (MAC)
PRP	pseudo-random permutation (PRP)
PRG0	pseudo-random generator (PRG)
h_{op}	a hash function to generate the key for op
g	a generator of a prime-order cyclic group $\mathcal{G}$ with order $\sim 2^{2k}$
R	routing segment, e.g., the ingress and egress ports
x_e, g^{x_e}	the private and public key pair of end host $e \in \{S, D\}$
EXP_i	expiration time for a packet at node n_i
FS	forwarding segment
s	a symmetric onion key shared with the sender
IV	per-packet initial vector
γ	per-hop MAC
β	the opaque component of a packet header
O	onion data packet

Table 5.2: Notation used in the TARANET protocol.

how to reach the next hop on the path. The FS is encrypted using a secret value known only to the router that created the FS. As shown in Section 5.3.4, these FSes are included in every data packet: each node can then the retrieve the FS it created, decrypt it, and recover the packet processing information within. Unlike HORNET, we do not store the expiration time EXP in a FS, but include it alongside the FS in the packet (see Section 5.3.3.2). This allows the sender to set a different expiration time for each packet and limit the time window in which the packet is valid, which is necessary for replay protection.

5.3.3 Data Packet Processing

5.3.3.1 Requirements

TARANET data packets are fixed-size onion packets whose integrity is protected by per-hop MAC. Processing these packets should satisfy the following three requirements:

- An output packet cannot be linked to the corresponding input packet without compromising the processing node's local secret value.
- Processing a packet cannot leak a node's position on the path.
- Processing a packet cannot change the packet size regardless of underlying operations.

The last requirement is particularly challenging to satisfy, since TARANET allows flow mutations. Consider the split operation, which takes a fixed-size packet and creates two uncorrelated packets of the same size. The splitting procedure needs to ensure that subsequent nodes can verify the MACs in both new packets.

5.3.3.2 Data Packet Format

TARANET data packets are shown in Figure 5.2. At the beginning of each packet is an IV field that carries a fresh initial vector for each packet in a flowlet. After the IV field are four fields that form an onion layer: an FS, a per-hop MAC, control bits, and the expiration time. The rest of the fields, including the rest of header information, padding bits, and the payload, are encrypted, and are thus opaque to the node that processes the packet.

When a packet arrives, the first three fields are accessible to a node without requiring cryptographic processing, so we call these fields as public state. The control bits and the expiration time are only available after the node decrypts the packet, so they are called secret state. In addition, each header is padded to a fixed size regardless of the actual number of nodes on the path, and the padding bits are inserted between the header and the payload.

5.3.3.3 TARANET Packet Creation

Both end hosts generate data packets using a subroutine shown in Algorithm 5. The subroutine creates an onion packet to be forwarded from node n_k to node n_l. For each onion layer, it computes a per-hop MAC (Line 14) and onion-encrypts both the header (Line 12) and the payload (Line 14).

One important feature of this onion encryption algorithm is to add per-hop state (specifically, an FS, a MAC, control bits and an expiration time) to the packet header without changing its total size. The

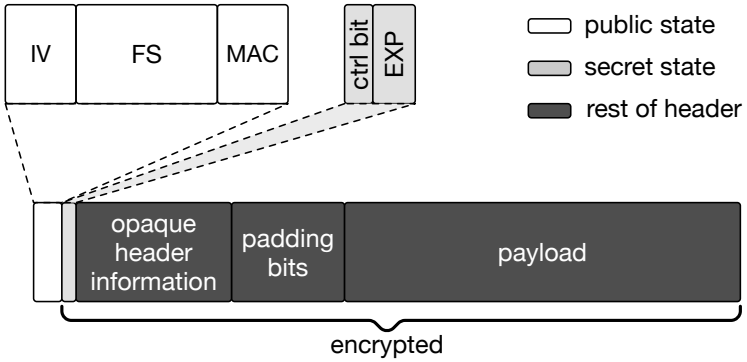


Figure 5.2: TARANET packet format.

function achieves this feature by strategically pre-computing the padding bits in the header (Line 6) to ensure that the trailing c bits of header after encryption are always equal to 0^c. As a result, the trailing zero bits can be truncated without losing information when the header is encrypted again (Line 12).

Algorithm 5 Create a partial data packet.

1: **procedure** CREATE_ONION_ROUTINE
   Input: $\{s_i\}$, $\{FS_i\}$, $\{ctrl_i\}$, $\{\text{EXP}_i\}$, IV, k, l, O ▷ with $k < l$
   Output: $(IV_k, FS_k, \gamma_k, \beta_k, O_k)$
2: $\phi_k \leftarrow \varepsilon$
3: $IV_k \leftarrow IV$
4: **for** $i \leftarrow k+1, \ldots, l$ **do**
5: $IV_i \leftarrow \mathsf{PRP}(h_{\mathsf{PRP}}(s); IV_{i-1})$
6: $\phi_i \leftarrow (\phi_{i-1} \,\|\, 0^c) \oplus$
$\mathsf{PRG0}(h_{\mathsf{PRG0}}(s_{i-1} \oplus IV_{i-1}))_{[(r-i-1)c+b..rc+b-1]}$
7: **end for**
8: $\beta_l \leftarrow \{\text{RAND}(c(r-l-1)) \,\|\, \phi_l\}$
9: $O_l \leftarrow \mathsf{ENC}(h_{\mathsf{ENC}}(s_l); IV_l; O)$
10: $\gamma_l \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s_l \,\|\, IV_l); FS_l \,\|\, \beta_l \,\|\, O_l)$
11: **for** $i \leftarrow (l-1), \ldots, k$ **do**
12: $\beta_i \leftarrow \Big\{ctrl_i \,\|\, \text{EXP}_i \,\|\, FS_{i+1} \,\|\, \gamma_{i+1} \,\|\, \beta_{i+1[0..c(r-2)-1]}\Big\}$
$\oplus\, \mathsf{PRG0}(h_{\mathsf{PRG0}}(s_i \,\|\, IV_i))_{[0..b+(r-1)c-1]}$
13: $\gamma_i \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s_i \,\|\, IV_i); FS_i \,\|\, \beta_i \,\|\, O_i)$
14: $O_i \leftarrow \mathsf{ENC}(h_{\mathsf{ENC}}(s_i); IV_i; O_{i+1})$
15: **end for**
16: **end procedure**

Normally, an end host creates a packet that traverses the whole path from the first node n_0^{dir} to the last node $n_{l^{dir}-1}^{dir}$. It generates such a packet by setting $k = 0$, $l = l^{dir} - 1$, and all $ctrl_i = \text{FWD}$ in function CREATE_ONION_ROUTINE.

Generate splittable packets Creating a data packet that can be split into two packet requires an end host to first create two children packets and then merge them into a single packet. Because we require all

packets to have the same size, i.e., both children packets have to be of the same size as their parent, the key challenge is to guarantee that the per-hop MACs in the children packets successfully verify even after the splitting node adds padding bits to the children packets. For this reason, the splitting node generates padding bits by a PRG keyed by the key shared with the end host, so that the end host can predict the padding bits and pre-compute the per-hop MACs in both resulting packets accordingly.

Algorithm 6 shows the function to create a splittable data packet. At a high level, CREATE_SPLITTABLE_DATA_PACKET invokes the

CREATE_ONION_ROUTINE three times: it first creates two children packets using CREATE_ONION_ROUTINE (Line 3 and 5), merges the resulting packets into a new payload (Line 6), and finally executes CREATE_ONION_ROUTINE again to generate the parent packet (Line 7). To ensure the correctness of the per-hop MACs in the children packets after the payloads are padded, the function generates the padding bits using a PRG keyed by the shared key between the end host and the splitting node so that the latter can re-generate the padding bits accordingly (Lines 3 and 4). After the MACs are computed for the children packets, the deterministic padding bits are truncated so that two children packets can fit into the payload of their parent packet.

Algorithm 6 Create a data packet that can be split into two new packets.

1: **procedure** CREATE_SPLITTABLE_DATA_PACKET
Input: $\{s_i\}$, $\{FS_i\}$, $\{ctrl_i\}$, $\{\text{EXP}_i\}$, IV, IV_0, IV_1 O_0, O_1, k
Output: $(IV_0, FS_0, \gamma_0, \beta_0, O_0)$
2: $O_0 \leftarrow O_0 \parallel \mathsf{PRG0}(h_{\mathsf{PRG0}}((s_k \oplus IV_0) \parallel \text{“left”}))_{[0..\frac{m}{2}+rc-1]}$
3: $(IV_0', FS_0', \gamma_0', \beta_0', O_0') \leftarrow$
CREATE_ONION_ROUTINE$(\{s_i, \forall i \geq k\}, \{FS_i, \forall i \geq k\},$
$\{\mathsf{FWD}\}, \{\text{EXP}_i, \forall i \geq k\}, IV_0, k, l_{dir} - 1, O_0)$
4: $O_1 \leftarrow O_1 \parallel \mathsf{PRG0}(h_{\mathsf{PRG0}}((s_k \parallel IV_1) \parallel \text{“right”}))_{[0..\frac{m}{2}+rc-1]}$
5: $(IV_1', FS_1', \gamma_1', \beta_1', O_1') \leftarrow$
CREATE_ONION_ROUTINE$(\{s_i, \forall i \geq k\},$
$\{FS_i, \forall i \geq k\}, \{\mathsf{FWD}\}, IV_1, k, l_{dir} - 1, O_1)$
6: $O' \leftarrow (IV_0', FS_0', \gamma_0', \beta_0', \{O_0'\}_{[0..\frac{m}{2}+rc-1]}) \parallel$
$(IV_1', FS_1', \gamma_1', \beta_1', \{O_1'\}_{[0..\frac{m}{2}+rc-1]})$
7: $(IV_0, FS_0, \gamma_0, \beta_0, O_0) \leftarrow$
CREATE_ONION_ROUTINE$(\{s_i, \forall i < k\},$
$\{FS_i, \forall i < k\}, \{\mathsf{FWD}, \ldots, \mathsf{FWD}, \mathsf{SPLIT}\},$
$\{\text{EXP}_i, \forall i < k\}, IV, 0, k - 1, O')$
8: **end procedure**

5.3.3.4 Onion Layer Removal

Nodes remove onion layers when processing data packets. It essentially reverses a single step of CREATE_ONION_ROUTINE. Algorithm 7 details this five step process. First, the intermediate node retrieves the symmetric onion key s shared with the sender (Line 3); second, the node verifies a per-hop MAC using a key derived from

s (Line 4); third, the node ensures that the packet's size remains unchanged by adding padding bits to the header and decrypting the resulting padded header with a stream cipher; fourth, the control bits are extracted (Line 6); finally, the payload is decrypted (Line 7) and the next initialization vector is obtained by applying a PRP keyed with s to the current IV (Line 8).

Note that the onion layer removal algorithm is different from a simple decryption in two ways. First, the size of the packet remains the same after processing, which prevents leaking information about the total number of hops between the sender and receiver. Second, the processing only happens at the head of the packet, which reveals no information about the processing node's position on the path.

Algorithm 7 Remove an onion layer.

1: **procedure** REMOVE_LAYER
   Input: P, SV
   Output: $ctrl$, P^o, R, EXP
2: $\{IV \parallel FS \parallel \gamma \parallel \beta \parallel O\} \leftarrow P$
3: $s \parallel R \leftarrow \mathsf{PRP}^{-1}(SV, FS)$
4: **check** $\gamma = \mathsf{MAC}(h_{\mathsf{MAC}}(s \parallel IV); FS \parallel \beta \parallel O)$
5: $\zeta \leftarrow \{\beta \parallel 0^c\} \oplus \mathsf{PRG0}(h_{\mathsf{PRG0}}(s \parallel IV))_{[0\ldots(r-1)c+b-1]}$
6: $ctrl \parallel \text{EXP} \parallel FS' \parallel \gamma' \parallel \beta' \leftarrow \zeta$
7: $O' \leftarrow \mathsf{DEC}(h_{\mathsf{DEC}}(s); IV; O)$
8: $IV' \leftarrow \mathsf{PRP}(h_{\mathsf{PRP}}(s); IV)$
9: $P^o \leftarrow \{IV' \parallel FS' \parallel \gamma' \parallel \beta' \parallel O'\}$
10: **end procedure**

Depending on the value of control bits $ctrl$, the intermediate node performs one of the following two actions: FWD, or SPLIT. A node can split a data packet into two new packets by Algorithm 8. First, the payload is split into two new packets (Line 2). Then the node pads both newly generated packets to the fixed size m using pseudo-random bits obtained from a PRG keyed by s (Line 3 and 4).

Algorithm 8 Split a data packet into two new packets.

1: **procedure** SPLIT_ONION_PACKET
   Input: O, s, IV
   Output: P_0^o, P_1^o
2: $\{P_0' \parallel P_1'\} \leftarrow O$
3: $P_0^o \leftarrow P_0' \parallel \mathsf{PRG0}(h_{\mathsf{PRG0}}((s \parallel IV) \parallel \text{“left”}))_{[0..\frac{m}{2}+rc-1]}$
4: $P_1^o \leftarrow P_1' \parallel \mathsf{PRG0}(h_{\mathsf{PRG0}}((s \parallel IV) \parallel \text{“right”}))_{[0..\frac{m}{2}+rc-1]}$
5: **end procedure**

5.3.4 Data Transmission Phase

5.3.4.1 End Host Processing

To send packets to receiver D, sender S first makes sure that the flowlet has not expired. Then S chooses a value EXP_{min}, which has to be larger than its local time plus the end-to-end forwarding delay plus the maximum global clock skew. We expect that adding 1 s to the local time would be adequate for most circumstances. However, S cannot set the packet expiration time to be equal at every hop, as otherwise this value could be used as common identifier (which violates the bit-pattern unlinkability property. Instead, S chooses an offset $\Delta_i \in [0, \Delta_{max}]$ uniformly at random, for each node n_i^f on the path. For every packet sent out, S determines EXP_{min} and computes $\text{EXP}_i = \text{EXP}_{min} + \Delta_i$ for each node. The value Δ needs to be chosen large enough to ensure that the interval $[\text{EXP}_{min}, \text{EXP}_{min} + \Delta]$ overlaps with the intervals of a large number of other concurrent flows. We expect that $\Delta \approx 5$ s would be a safe choice.

After determining $\{\text{EXP}_i\}$, S also needs to decide which flow mutation actions the packet will adopt. In case of packet splitting, S also needs to decide where to split the packet. For a packet that is forwarded to the receiver without being split, we denote the payload to send is O. For a packet that is split, we denote the payloads of the children packets as O_0 and O_1. Let k be the index of the node where the packet is split. Accordingly, $ctrl_i = \text{FWD}, \forall i \neq k$. Third, S uses s_{SD} to encrypt the payload. This end-to-end encryption prevents the last hop node from obtaining information about the data payload. S also generates a unique nonce IV for the packet. If the packet is splittable, S generates another two unique nonces IV_0 and IV_1. Fourth, if the packet will be split, S creates the packet P by

$$P = \text{CREATE\_SPLITTABLE\_DATA\_PACKET}(\{s_i^f\}, \{FS_i^f\}, \{ctrl_i^f\}, \{\text{EXP}_i^f\}, IV, IV_0, IV_1, O_0, O_1, k) \quad (5.1)$$

If the packet will only be forwarded to the receiver without a splitting action, S creates the packet P by

$$P = \text{CREATE\_ONION\_ROUTINE}(\{s_i^f\}, \{FS_i^f\}, \{ctrl_i^f\}, \{\text{EXP}_i^f\}, IV, 0, l^f - 1, O) \quad (5.2)$$

Finally, S forwards P to the first hop node towards the receiver.

The process by which D sends packets back to S is similar to the above procedure, but D will use the forwarding segments and onion keys for the backward path. However, right after S finishes the setup phase, D has not yet obtained g^{x_S}, $\{s_i^b\}$, nor $\{FS_i^b\}$. In the TARANET data transmission phase, the first packet that S sends to D includes x_S, $\{s_i^b\}$ and $\{FS_i^b\}$ as the payload.

When an end host (S or D) receives a data packet P, it can retrieve the data payload O from the packet by $O = P_{[rc..rc+m-1]}$ The resulting O can thus be decrypted by s_{SD} to retrieve the plaintext payload.

5.3.4.2 Intermediate Node Processing

When a node receives a data packet $P = (IV, FS, \gamma, \beta, O)$, with the local secret SV, it first removes an onion layer by

$$ctrl, P^o, R, \textsc{exp} = \textsc{remove\_onion\_layer}(P, SV) \tag{5.3}$$

Note that the MAC must check in REMOVE_ONION_LAYER for the process to move on. Otherwise, the node simply drops the packet. Then, the node checks $t_{curr} < \textsc{exp}$ and ensures that the flowlet has not expired. Afterwards, the node checks the control bits belonging to the current hop. If $ctrl =$ SPLIT, the resulting payload P^o must contain two sub packets. The node creates two children packets P_0^o, P_1^o:

$$\{P_0^o, P_1^o\} = \textsc{split\_onion\_packet}(O, s, IV) \tag{5.4}$$

Lastly, if the packet is not dropped, the node forwards the resulting packet according to the routing decision R.

5.4 Security Analysis

We discuss TARANET's defenses against passive (Section 5.4.1) and active attacks (Section 5.4.2). We also conduct a quantitative analysis of TARANET's anonymity-set size using the Internet topology and real-world packet traces (Section 5.4.3). Our result shows that TARANET's anonymity set is 4 to 2^{18} times larger than those of LAP and Dovetail. We present a formal proof that the TARANET protocol conforms to an ideal onion routing protocol as defined by Camenisch and Lysyanskaya [51] in our technical report [59].

5.4.1 Defense against Passive Attacks

Flow dynamics matching In flow-dynamics matching attacks [66, 134], adversarial nodes can collude to match two observed flows by their dynamics, such as transmission rate. TARANET prevents such attacks by normalizing the outgoing transmission rate of all flowlets through the use of chaff traffic. Adversarial nodes are unable to distinguish chaff traffic from real traffic. Accordingly, no flow dynamics are available to the adversary to perform matching.

Template attacks TARANET enables end hosts to shape their traffic by adding chaff packets to hide their real traffic patterns. The resulting traffic pattern of an outgoing flowlet is uniform across the network. In addition, all TARANET packets have the same length, preventing information leakage from packet length. The combination of these two features completely neutralizes template attacks.

Network statistics correlation These attacks rely on the capability of the adversary to observe macroscopic flow characteristics which leak de-anonymizing information. Because of the uniformity of flowlets, no such information is leaked in TARANET for isolated unidirectional flows. However, if the attacker is able to link the flowlets corresponding to a bidirectional flow by their starting or ending time, then an attack based on the RTT could still be possible. Such an attack can be thwarted by adding delays for setup packets and flowlet start at the receiver, according to the path length (the shorter the path, the longer the delay), as suggested by previous work [57, Section 5.1].

5.4.2 Defense against Active Attacks

Tagging attacks A compromised node can modify packets adding tags that are recognizable by downstream colluding nodes. This enables flow matching across flows observed at different nodes [148]. TARANET defends against such attacks through its per-hop packet authenticity (see Section 5.3). A benign node will detect and drop any packet that has been modified.

Clogging attacks In clogging attacks, an adversary intentionally causes network congestion [84, 133], or fluctuation [53] to create jamming or noticeable network jitter on relay nodes, and match such patterns to deanonymize the path. Different from throughput fingerprint attacks that aim to exert no influence on existing traffic patterns, clogging attacks aggressively change the traffic patterns on victim links and are prone to detection. First, clogging attacks in TARANET itself require DDoS capabilities because of nodes' high bandwidth within the network. In addition, TARANET nodes attacked by clogging would run out of cached chaff packets, which in turn shuts down the flowlet and prevents any additional matching. Moreover, given the large number of flowlets in the network at any given time, the number of flowlets terminated due to normal operations is large, which hides the fact that the specific attacked flowlet is terminated.

Flow dynamics modification attacks Traffic pattern modulation attacks require attackers to modulate inter-packet intervals to either create recognizable patterns (e.g., flow watermarking attacks [102, 104]), or embed identity information (e.g. flow fingerprinting attacks [103]), so that downstream adversarial nodes can deanonymize traffic by extracting the introduced traffic patterns. Depending on the amount of perturbation introduced by the adversary, we can distinguish two cases. In the first one, the adversarial actions fail to exhaust the cached chaff packets on the node under attack for the target flowlet. In this case, the outgoing rate for the flowlet at the node remains unchanged, and the attack is ineffective. In the

second case, the victim node runs out of cached chaff packets for the target flowlet. In this case, the node terminates the flowlet to prevent downstream nodes from observing the injected patterns.

5.4.3 Anonymity-set Size Evaluation

Relationship anonymity set Network-layer anonymity protocols are vulnerable to passive attacks based on network topology information launched by a single compromised AS. Compared to overlay-based anonymity systems [79] that allows global re-routing, traffic of network-layer anonymity protocols follows paths created by underlying network architectures. By observing the incoming and outgoing links of a packet, a compromised AS can derive network location information of communicating end hosts. For example, in Figure 5.3, by forwarding a packet from AS1 to AS3, AS2 knows that the sender must reside within the set {AS0, AS1} and the receiver falls into the set {AS3, AS4, AS5}. We name the former anonymity set *sender anonymity set*, denoted as S_s, and call the latter anonymity set *recipient anonymity set*, denoted as S_d. Accordingly, we define *relationship anonymity set* $S_r = \{(s, d)|s \in S_s,\ d \in S_d\}$.

To evaluate relationship anonymity of different protocols, we use anonymity-set size as the metric. By definition of S_r, the anonymity-set size $|S_r| = |S_s| \times |S_d|$. In Figure 5.3, there are 8 hosts in both AS0 and AS1. Thus, $|S_s| = 16$. Similarly, we can calculate that $|S_d| = 24$ and $|S_r| = 16 \times 24 = 384$.

Protocol designs influence corresponding anonymity-set sizes. In LAP and Dovetail, by analyzing header formats, a passive adversary can determine its position on the packet's path, i.e., its distances from the sender and the receiver [105, 156]. In Figure 5.3, if the adversary in AS2 knows the sender is 2 hops away and the receiver is 1 hops away through analyzing packet headers, it can deduce that the sender must be in AS0 and the receiver must be in AS3. The resulting anonymity-set size is reduced to 8 * 8 = 64. In comparison, TARANET and HORNET's header designs prevent their headers from leaking position information to adversaries.

Experiment setup We use a trace-based simulation to evaluate anonymity set sizes of different network-layer anonymity protocols in real world scenarios. We obtain the real-world AS-level topology from CAIDA AS relationship dataset [2]. We also annotate each AS with its IPv4 address space using the Routeview dataset [18]. In addition, we estimate real-world paths using iPlane traceroute datasets [12]. We use the traceroute traces on Dec. 12th, 2014. For each IP address–based trace, we convert it to AS path. Our preliminary analysis shows that the median AS path length is 4 and the average AS path length is 4.2. More than 99.99% of AS paths have length less than 8.

For each AS on a path in our path dataset, we compute the sizes of the relationship anonymity sets observed by the compromised AS in one of two scenarios: 1) the AS knows its position on path as in LAP

and Dovetail; 2) the AS has no information about its position on the path as in HORNET and TARANET. To compute anonymity set sizes, we first derive relationship anonymity sets composed by ASes. Then we compute the number of hosts in the ASes as the size of anonymity set size. We approximate the number of hosts within an AS by the number of IPv4 addresses belonging to that AS.

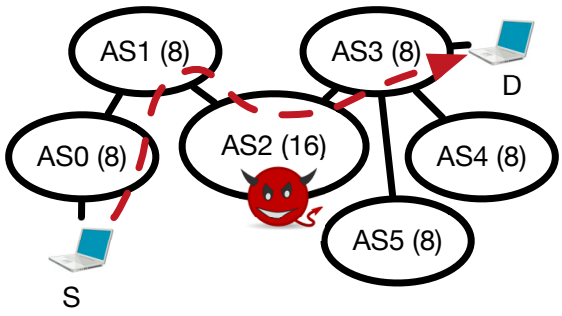
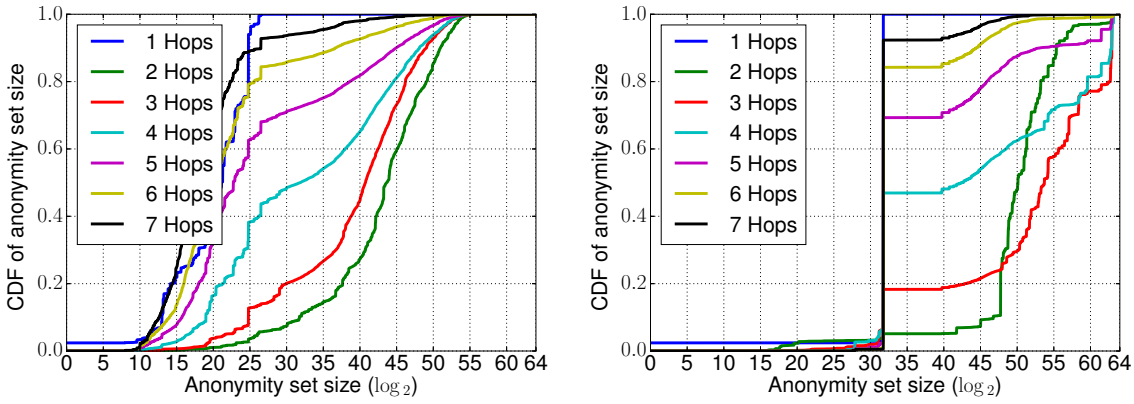


Figure 5.3: a) A toy example of an adversary that exploits topology information to de-anonymize a flowlet between sender S and receiver D. ASx (y) denotes an AS with AS number x and y hosts attached. We assume that the adversary compromised AS2.

Result Figure 5.4(a) demonstrates CDFs of anonymity-set sizes for LAP and Dovetail observed by a compromised AS. Figure 5.4(b) shows the CDF of anonymity-set sizes for TARANET and HORNET. In general, anonymity-set sizes of TARANET and HORNET exceed 2^{32} with probability larger than 95% regardless of the adversary's on-path positions. The 90th percentiles of anonymity-set sizes of TARANET and HORNET are 4–2^{18} times larger than those of LAP and Dovetail depending on the distances between senders and receivers. We remark that when an AS is 6 or 7 hops away from a sender, it is the last-hop



(a) LAP and Dovetail (b) TARANET and HORNET

Figure 5.4: a) Cumulative Distribution Functions (CDF) of anonymity set sizes for LAP and Dovetail. b) CDFs of anonymity set sizes for TARANET and HORNET. In both a) and b), different lines demonstrate anonymity set size distribution observed by an adversary that is a fixed number of AS hops away from S.

AS with high probability, because 99.99% paths are less than 8 hops long. When the compromised ASes are 1 hop away from senders and when the ASes are close to receivers (6–7 hops away from senders), the gap between TARANET/HORNET and LAP/Dovetail is largest.

Topology-based attacks and traffic analysis In LAP, Dovetail, and HORNET, when an adversary compromises more than 1 AS on a path, he/she can correlate observation from different non-adjacent ASes by traffic analysis, such as flow fingerprint attacks [103], to facilitate topology-based attacks. Assume that an adversary compromises q ASes and observes a series of sender anonymity sets $\{S_s^i; i \in [1,q]\}$ and a series of recipient anonymity sets $\{S_d^i; i \in [1,q]\}$. The resulting relationship anonymity set size $|S_r| = \min_{i\in[1,q]} |S_s^i| \times \min_{i\in[1,q]} |S_d^i|$. For example, in Figure 5.3, if the adversary compromises AS0 besides AS2 and uses traffic analysis to identify traffic from the same flowlet, the resulting relationship anonymity-set size $|S_r|$ is only 24 (1×24) compared to 384 when only AS2 is compromised.

TARANET improves over LAP, Dovetail, and HORNET by introducing defense against traffic analysis (see Section 5.4.1 and 5.4.2). By defeating traffic analysis and preventing correlation of flowlets at multiple non-adjacent ASes, TARANET enlarges the observed relationship anonymity set size. The resulting relationship anonymity set is only the smallest one among the relationship anonymity sets observed by non-collaborative compromised ASes. $|S_r| = \min_{i\in[1,q]} |S_s^i| \times |S_d^i|$. For instance, when the adversary compromises AS0 besides AS2 and uses traffic analysis to correlate observed flowlets, the resulting relationship anonymity-set size $|S_r|$ increased to 56.

5.5 Evaluation

This section describes our implementation of TARANET, a performance evaluation,and our evaluation of bandwidth overhead added by end-to-end traffic shaping.

5.5.1 Implementation on High-speed Routers

We implement TARANET's setup and data transmission logic on a software router. We use Intel's Data Plane Development Kit (DPDK [8], version 2.1.0), which supports fast packet processing in user space. We assemble a customized cryptography library based on the Intel AESNI sample library [10], and the curve25519-donna [7] and PolarSSL [17] libraries. We use 128-bit AES counter mode for encryption and 128-bit AES CBC-MAC.

5.5.2 Performance Evaluation

Our testbed is composed of a commodity Intel server and a Spirent TestCenter packet generator [20]. The Intel server functions as a software router and is equipped with an Intel Xeon E5-2560 CPU (2.70 GHz, 2 sockets, 8 cores per socket) and 64 GB DRAM. The server also has 3 Intel 82599ES network cards with 4 ports per card, and is connected to the packet generator through twelve 10-Gbps links. Thus the testbed is capable of testing throughput up to 120 Gbps.

To remove implementation bias and allow fair comparison with other anonymity protocols, we additionally implement LAP [105], Dovetail [156], HORNET [57], and Sphinx [71] logic using our custom cryptography library and DPDK. Note that LAP, Dovetail, and HORNET are high-speed network-layer anonymity protocols but cannot defend against traffic analysis attacks. Sphinx is a mix network that requires performing public key cryptographic operations for every data packet and thus incurs high computation overhead.

We expect that TARANET's performance is lower than LAP, Dovetail, and HORNET, because TARANET's traffic-analysis resistance property incurs additional overhead by design, and that TARANET should outperform Sphinx, which also offers traffic-analysis resistance. Additionally, as an essential requirement of high-speed network-layer anonymity protocol, we expect that TARANET should be capable of supporting high forwarding throughput.

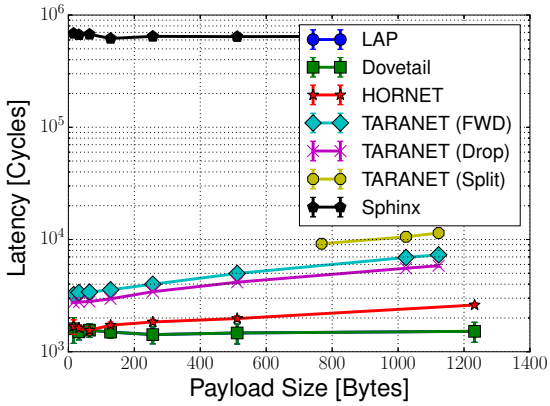
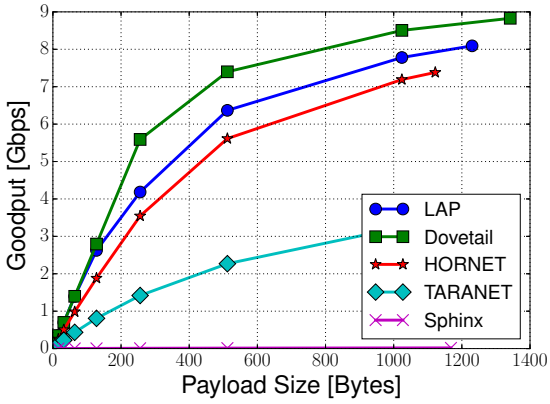
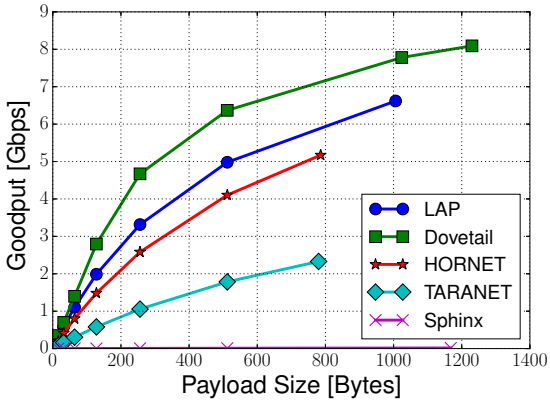


Figure 5.5: a) Average latency of processing a packet for different protocols with error bars (95% confidence intervals). For a packet with "SPLIT" flag, because the payload has to contain at least two other packet headers, we only test packets with payloads at least 768 bytes. Lower is better.

Processing latency We first evaluate the average latency of processing a data packet on a single core using different anonymity protocols. For TARANET, we also compare the latency of performing different mutation actions according to the control flag in the packet. The results are shown in Figure 5.5.



(a) Goodput: 7-hop header



(b) Goodput 14-hop header

Figure 5.6: a) Data forwarding goodput on a 10 Gbps link for packets with 7-hop headers and different payload sizes; b) data forwarding goodput on a 10 Gbps link for packets with 14-hop headers and different payload sizes. Higher is better.

TARANET's processing latency is comparable to LAP, Dovetail, and HORNET. When the payload size is smaller than 64 bytes, processing a TARANET data packet (following the steps described above) incurs less than 1μs (≈3700 cycles) per-hop overhead on a single core. For payloads larger than 1024 bytes, the latency increases to up to 2μs (≈7200 cycles). Splitting a TARANET packet incurs only additional 1μs (≈4200 cycles). Since the total number of ASes on a path is usually less than 7 [57], TARANET processing will typically add only ∼20μs to the total end-to-end latency.

Processing a setup packet on our test machine incurs around 250μs (0.66M cycles) per hop per packet. This is due to setup packets requiring a DH key-exchange operation. However, for path lengths of less than 7 hops, this latency adds less than 2ms to the first packet at the start of each flowlet.

Goodput Goodput measures the throughput of useful data that can be transmitted by the protocol as it separates data throughput and packet header overhead. Figures 5.6(a) and 5.6(b) show the goodput of different protocols with 7-hop and 14-hop headers, respectively, on a single 10 Gbps link with 1 core assigned. We observe that even with longer processing latency and larger headers, TARANET still achieves ≈45% of HORNET's goodput in both cases. With a single core, TARANET can still achieve ∼0.37 Mpkt/s.

Maximum total throughput To evaluate the maximum total throughput of our protocol with respect to the number of cores, we test TARANET with all twelve 10 Gbps ports enabled while using all 16 CPU cores. Each port has 1 input queue and 1 output queue. To fully distribute the computation power of 16 cores to 12 input and 12 output queues, we assign 8 cores exclusively to 8 input queues, 4 cores each to one input and one output queue, and the remaining 4 cores to 8 output queues. The packet generator

generates packets that have random egress ports and saturate all 12 ports. Our evaluation finds that TARANET can process anonymous traffic at 50.96 Gbps on our software router for packets with 512 bytes of payload, which is comparable to the switching capacity of a commercial edge router [5].

Delay of flowlet setup For the setup phase, TARANET uses packet batching and randomization to protect against traffic analysis. Our observation is that if the number of flowlet setups is sufficient large, batching setup packets can still end up yielding a short setup delay.

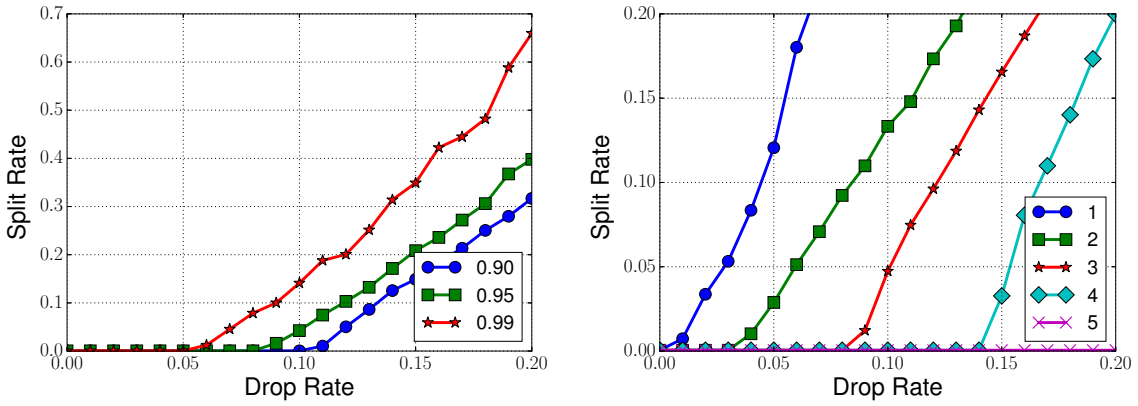
We conduct a trace-driven simulation using the CAIDA's anonymized packet traces to evaluate the setup phase's delay. The packet traces are recorded by the "equinix-chicago" monitor on a Tier-1 ISP's 10 Gbps link between 1-2 pm on Mar. 20th, 2014 [3] We assume the first packet in each flow in the dataset is a flowlet setup packet. We simulate the latency introduced by batching, randomizing, and cryptographic processing by injecting the setup packet trace into a TARANET node and varying the batch size.

The resulting latency of flowlet setups increases almost linearly as the batch sizes increases. When the batch size is 16, the per-hop latency is 1.6 $\pm$ 1.0 ms (95% confidence interval). When the batch sizes reaches 128, the per-hop latency increases up to 12 $\pm$ 7 ms. For a path with 7 AS-hops that means the flowlet setup will introduce less than $\sim$170 ms additional round-trip latency for the setup phase, which is a small proportion of the delay of an inter-continental path.

5.5.3 Overhead Evaluation

We conduct a trace-based evaluation of TARANET to evaluate added bandwidth overhead for end hosts and the amount of state for routers. For bandwidth overhead, we evaluate the number of splittable packets required to accommodate different levels of packet drops and the number of chaff traffic needed to shape real-world traffic. We use CAIDA's anonymized packet traces as discussed in Section 5.5.2. We filter away ICMP packets and small flows that has the size smaller than 10 packets or has the transmission rate lower than 1 byte/second.

Split rate First, we evaluate the number of splittable packets needed to account for packet drops. Note that an insufficient split rate causes a node to deplete its locally cached chaff and prematurely terminate a flowlet when there exist large number of packet drops. We convert each flow in the trace into flowlets where $B = 10$ kbps and $T = 1$ min and run the trace with different per-hop split rates, different drop rates, and various failure counters H. We set $L_{chf} = 3$ to let a node to cache 3 chaff packets at maximum for each flowlet. Figure 5.7(a) shows the required per-hop split rates regarding different drop rates to achieve different success rates. The observed drop rate in the Internet is around 0.2% [171]. For such a



(a) Split Rate: Different Success Rate

(b) Split Rate: Different H

Figure 5.7: a) Probability of splittable packets to guarantee different success rate. Failure counter $H = 2$. Different lines stand for different success rates. b) Probability of splittable packets for different parameter H (See Section 5.2.2) to achieve a 95% success rate.

low drop rate and a failure counter $H = 2$, a node can set the per-hop split rate to almost 0 and still obtain high success rates as much as 99%. To account for a highly lossy network link or an adversary that manipulates timing pattern through dropping packets, an end host can adjust the split rate of that node up to 5% even for a very high per-hop drop rate 10% to achieve 95% success rates.

Figure 5.7(b) demonstrates the required per-hop split rates with respect to different drop rates to guarantee a 95% success rate when the failure counter H ranges from 1 to 5. In general, given a certain packet drop rate, a larger H helps reduce the per-hop split rate. For instance, when H=1, the per-hop split rate can be as high as 12% for a packet drop rate of 5%. However, when H increases to 2, the required per-hop split rate is already at 3.4%. When we lets H=4, we can accommodate a per-hop drop rate of 15% by a per-hop split rate as small as 3.7%. We remark that when flowlets have smaller bandwidth B (as will be shown next), the success rate increases.

Chaff overhead We then evaluate the bandwidth overhead of the added chaff traffic. We convert real-world flows in the CAIDA's packet traces into flowlets and compute the amount of chaff required. Note that we normalize the resulting overhead normalize the chaff overhead through dividing it by the total traffic volumes, in order to remove the impact of traffic volumes and network sizes.

Figure 5.8 graphs the chaff overhead needed for the conversion when the bandwidth parameter of flowlets B varies and T=1 min. Generally, large B results in large chaff overhead. When we use B=5 kbps, the overhead of chaff packets is 7%. In comparison, when B becomes 20 kbps, the overhead of chaff traffic is increased to 31%. Moreover, we observe that small flow sizes, such as UDP flows for DNS lookups, lead

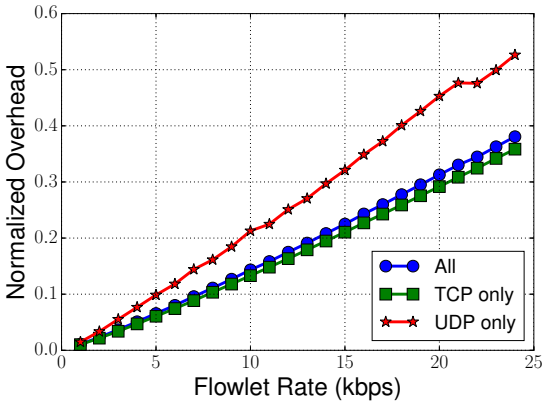


Figure 5.8: Bandwidth overhead caused by added chaff traffic for traffic shaping. We normalize the overhead through dividing the added overhead by the original bandwidth. Lower is better.

to large chaff overhead given the same B because more packets in the resulting flowlets are chaff packets. In Figure 5.8, the chaff overhead for UDP flows are larger than TCP flows because the size of UDP flows are usually larger than TCP flows.

Required amount of state and scalability To enable traffic shaping for flowlets, an intermediate node maintains state bounded by the node's bandwidth (Section 5.2.2). To demonstrate the scalability of TARANET with respect to the Internet traffic volumes, we evaluate the amount of state required for a node to process real Internet traffic.

For each flowlet that consumes bandwidth B, a node maintains L_{chf} chaff packets and a failure counter required by the end-to-end traffic shaping technique (Section 5.2.2). We set $L_{chf} = 3$ and vary the bandwidth parameter B for flowlets. Using the flowlets converted from our CAIDA flow trace, we can evaluate the amount of state required. Our results show that a node stores 90 MB state when B=10 kbps for a 10 Gbps link and that the node needs to store 52 MB state when B=20 kbps.

Chapter 6

Discussion

This chapter aims to address real-world issues of the proposed anonymity systems. Specifically, Section 6.1 discusses the incentives and strategies of incrementally deploying infrastructure-based anonymity systems. Section 6.2 presents necessary infrastructures to disseminate path information anonymously in FIAs, which is a prerequisite for operating the proposed anonymity systems. Section 6.3 addresses the integration of network-layer anonymity systems with security mechanisms of other layers to offer higher anonymity guarantees. Section 6.4 discusses two different methods to establish sessions and why the proposed anonymity systems adopt the recursive setup instead of the telescopic setup. Finally, Section 6.5 presents attacks that are out of the scope of this dissertation and proposes possible defenses.

6.1 Incremental Deployment

Deployment Incentives We envision that ISPs have incentives to deploy anonymous communication systems to offer strong anonymity as a service to their privacy-sensitive users and corporations, or other customer ISPs who in turn desire to offer anonymous communication services. Deploying anonymity services would give the ISPs a competitive advantage: both private and business customers who want to use an anonymity service would choose an ISP that offers privacy protection.

Incremental Deployment Strategy The minimal requirement for deploying the proposed infrastructure-based anonymity systems includes a topology server that distributes path information, a few ISPs that deploy border routers supporting the anonymity protocol, and end hosts that run corresponding client software. We remark that the network architectures that we consider, such as NIRA [195], NEBULA [35], and SCION [198], already assume such topology servers as part of their control-plane infrastructure.

Admittedly, more ISPs that deploy the infrastructure-based anonymity systems would increase the

anonymity set size which in turn benefits all users. However, a few initial ISPs that deploy the anonymity systems but share no physical links can establish tunnels between each other through legacy ISPs and start to carry anonymous traffic among users. As more ISPs deploy the system and join the network, tunnels are increasingly replaced with direct ISP-to-ISP connections, which provides increasingly better guarantees.

6.2 Retrieving Path Information Anonymously in FIAs

Both HORNET and TARANET assume that the source can obtain a forward path and a backward path to an intended destination anonymously in FIAs. We briefly discuss how a source host can retrieve two such paths in NIRA, SCION and Pathlets.

SCION hosts rely on path servers to retrieve paths. In SCION, each destination node registers on a central server its two "half" path: an up path from the destination node to the network "core" and a down path the path in the reverse direction. To compose full paths (forward and backward paths) between a source and a destination, the source only needs to anonymously fetch the destination's up path and down path and combine them with its own half paths.

To anonymously retrieve a destination's half paths, the source can use one of the following two methods. As a first method, the source can obtain the path to/from a path server through an unprotected query using other schemes, from resolver configuration, or from local services similar to DHCP. The source then establishes a HORNET or a TARANET session to the server. Once a session is created, the source can proceed to anonymously request half paths of the destination. Though it is possible to reuse the established session to a path server to query multiple paths (for different destinations) for better efficiency, using a separate session to retrieve each path is more secure because it prevents profiling attacks.

Alternatively, the source can leverage a private information retrieval (PIR) scheme [61] to retrieve the path anonymously from the path server, so that the path server cannot distinguish which destination the source connects to. However, a PIR scheme will inevitably add bandwidth and computational overhead to both the source and the path server, increasing session setup phase latency [128].

In NIRA and Pathlets, the situation is different because routing information (i.e., inter-domain addresses and route segments, and pathlets, respectively) is disseminated to users. The source can thus keep a local path database, querying it (locally) on demand.

6.3 Integrating with Security Mechanisms at Different Layers

At the network layer, infrastructure-based anonymity systems can benefit from ASes that offer traffic redirection to mitigate topology-based attacks. For instance, ASes can allow paths that deviate from the valley-freeness policy to increase the anonymity set size of end hosts. This enables a trade-off between path length and anonymity, as described by Sankey and Wright [156].

Also, upper-layer anonymity protocols can be used in conjunction with the proposed systems to provide stronger anonymity guarantees. For example, to entirely remove the concerns of topology-based attacks, a single-hop proxy or virtual private network (VPN) could be used to increase the size of the anonymity sets of end hosts. Similar solutions could also protect against upper-layer de-anonymization attacks, in particular, fingerprinting attacks on the transport protocol [165].

At lower layers, PHI and HORNET are also compatible with link-layer protection such as link-level encryption. Link encryption prevents an adversary from eavesdropping on a link from being able to distinguish individual sessions from each other, therefore making confirmation attacks harder for this type of adversary.

6.4 Recursive vs. Telescopic Session Setup

Both HORNET and TARANET use a single round of setup messages to negotiate shared keys with intermediate nodes in a recursive manner. Other systems (e.g., Tor) use a telescopic setup. In the telescopic method, a source iteratively sets up a shared key with each AS: the source sets up a shared key with the first-hop AS; the source sets up a shared key with the n-th-hop AS through the channel through 1st-hop AS to $(n-1)$-th-hop AS.

The recursive setup is advantageous because it incurs lower setup latency and adapts to asymmetric paths. Regarding latency, the recursive setup introduces $O(n)$ latency where n is the number of nodes on the path in comparison with $O(n^2)$ latency required by the telescopic setup. Regarding the adaptability to irreversible paths, the recursive setup can function correctly when the forward path and the backward path are asymmetric. However, the telescopic setup requires that all paths must be reversible, which does not hold in the Internet [100] and the FIAs [35, 142].

A drawback of the recursive setup is that it does not provide perfect forward secrecy for the link between communicating parties in both HORNET and TARANET. Lacking perfect forward secrecy means that an adversary could record the observed traffic (in the setup phases, in particular), and if it later compromises a node, it learns which node was next on the path for each recorded session. Lacking perfect forward secrecy is an unavoidable limitation of having a setup that consists of a single round trip.

In comparison, the telescopic setup achieves perfect forward secrecy at the cost of diminished performance and adaptability.

It is important to note that in both HORNET and TARANET it is still possible to achieve perfect forward secrecy for the communication content, i.e., for the data exchanged between sources and destinations. The destination needs to generate an ephemeral Diffie-Hellman key pair, and derive an additional shared key from it. Destinations also need to generate a new local SV frequently, so in the event of a destination being compromised, it is not possible for the adversary to decrypt FSes used in expired sessions.

6.5 Limitations

Long-term intersection attack An adversary who observes the presence of all sender and receiver clients over a long period can perform intersection analysis [74, 122] to reveal pairs of clients that are repeatedly online during the same period. Clients can minimize their risk by being online not only when they are actively communicating, but in general, this attack is difficult to defend against through technical means. For further defense, PHI, HORNET, and TARANET all could be enhanced using existing solutions, such as dummy connections [44], or with the Buddies system [194], which allows clients to control which subset of pseudonyms appears online for a particular session. We leave analysis and evaluation of integration with such systems to future work.

Routing attacks Infrastructure-based anonymity systems rely on underlying network architectures for routing packets. Adversarial nodes can attack underlying network architectures to place themselves at strategic positions to launch traffic analysis [170, 182]. Although defeating routing attacks itself is beyond the scope of this dissertation, the network architecture candidates we consider offer control and integrity of packets' paths that prevent routing attacks. For example, SCION [198] and NEBULA [35] both embed integrity tags within paths to prevent path modification. Pathlet [92] pushes path selection to end hosts, enabling end hosts to select the path the packet traverses.

Denial-of-Service attacks An adversary can initiate a high volume of sessions (flowlets) passing through a node, to exhaust the node's computation power, bandwidth, and memory. Our systems cannot defend against such a DoS attack. To mitigate the DoS attack on a node's computation and bandwidth, the node can require sessions initiators to solve cryptographic puzzles [26]. Additionally, an ISP that offers anonymity service can also directly restrict the average session-initiation rate of its customers. We note that a DoS attack aiming to exhaust memory state by initiating a large number of sessions will fail because PHI

and HORNET nodes maintain no per-session state and because the amount of state that a node maintains is strictly linear to the node's actual bandwidth in TARANET. If the adversary in TARANET is able to accumulate sufficient bandwidth, such an attack will jam the network link of the victim node and become a bandwidth-based DoS attack.

Chapter 7

Related Work

This chapter reviews the existing anonymous communication systems. Given the abundance of the related work on anonymity systems, we focus on the most promising ones. Based on the underlying mechanisms and offered quality of service, we categorize the related work into four categories:

- Section 7.1 introduces the family of the Mix networks including the underlying mechanisms, the attacks, and the defenses.
- Section 7.2 describes the evolution of the onion routing systems. Specifically, we present the idea, the security/performance issues, and the solutions of Tor, the most popular anonymous communication system today.
- Section 7.3 discusses the basic concepts and issues of different peer-to-peer anonymous communication systems.
- Section 7.4 concentrates on a broad category of recent anonymous communication systems that feature provable strong anonymity and are suitable for anonymous messaging or micro-blogging.

7.1 Classical Mix Networks

7.1.1 Chaum's Mix Network

Chaum pioneered in the problem of anonymous communication and proposed to route messages through a sequence of "mix" servers that conceal the connection between input and output messages [56]. Each mix server performs two major functions: encrypting/decrypting each input message and changing the order among messages. Correspondingly, Chaum's Mix network design features two major components: 1) the mix message format based on layered RSA encryption/decryption and 2) the message mixing that

groups input messages into batches and randomizes the order within each batch. Additionally, Chaum's Mix network enables a reply scheme that allows the receiver to reply using a special cryptographic data structure called "return address" composed by the sender. Finally, Chaum also investigated the structure of a Mix network. He proposed a cascade structure where each message traverses all mixes in the same order and a free route structure where each message traverses only a sequence of mixes appointed by the sender.

However, neither the mix message format nor the mixing strategy in the original design is ideal. Regarding the mix message format, the RSA-based mix message format incurs substantial bandwidth overhead. Moreover, the design of the message format is based on heuristic security arguments rather than formal security proof. In fact, the original scheme fails to offer bitwise unlinkability as Pfitzmann and Pfitzmann demonstrated that the RSA-based message format leaks information when tagging attacks are present [146].

Regarding the mixing strategy, Chaum's mixing strategy is vulnerable to a powerful active attack called $n-1$ attack, where the attacker sends one victim message to an empty mix along with adversary-controlled messages to form a batch. When the mix flushes the batch, the attacker can recognize the victim message, breaking the anonymity guarantee. A more generalized version of the $n-1$ attack is called the blending attack [159]. Furthermore, batching can introduce significant delays. Thus, the initial Mix network is only suitable for delay-tolerant applications such as remailers. For example, remailers based on Chaum's Mix network model such as Babel [98], MixMaster [131], and Mixminion [70], have since been implemented and deployed.

7.1.2 Secure and Efficient Mix Message Formats

Mix message formats are complex cryptographic data structures that are difficult to reason about correctness, and the traditional designs rely on heuristic security arguments [56,70,72,131], leading to insecure designs. For example, Pfitzmann and Pfitzmann showed that Chaum's initial design is vulnerable to active attacks [146]. Minx [72] leaks information that allows an adversary to recover the full payload [164].

Recent research focuses on designs that have provable security guarantees [51,71,130,164]. Möller [130] first proposed a provably-secure message format. However, the proposed model is flawed because a message format conforming to the model can still leak the distance between an intermediate mix to the destination [51]. Camenisch and Lysyanskaya formally defined a secure mix message format, but the formal message format fails to consider return addresses for backward paths [51]. In comparison, Shimshock et al. [164] and Danezis and Goldberg [71] proposed provably secure mix message formats

that support bidirectional anonymous communication.

Another critical aspect of mix message format designs is bandwidth efficiency. The mix message formats of Mixminion [70] and Minx [72] require at least a full RSA cyphertext per hop. The provable designs proposed by Möller [130], Camenisch and Lysyanskaya [51], and Shimshock et al. [164] need multiple RSA ciphertexts per hop. The resulting header of a mix message becomes kilobytes long, which adds a significant bandwidth overhead to short messages. Sphinx [71] is the first mix message format that only takes 64 bytes per hop for 128-bit security levels by using Dan Bernstein's Curve25519 elliptic curve cryptographic system [42], reducing the bandwidth overhead by one order of magnitude. Additionally, the small header size of Sphinx also fits a network-layer packet, which is of $\sim$1500 Bytes. As a result, both HORNET and TARANET use Sphinx as the packet format for the setup phase for its provable security and low bandwidth overhead.

7.1.3 Defeating Blending Attacks

Another line of research about Mix networks is to defeat the blending attacks [159]. Traditional batching-based Mix networks are vulnerable to the blending attack, as demonstrated by O'Connor [137]. Diaz and Serjantov proposed to randomize the number of output messages as a binomial distribution of the cached input messages [77]. Diaz and Preneel demonstrated that the combination of the binomial mixing and chaff traffic guarantees a lower bound of anonymity for victim messages [76].

Other proposals discard using batching and instead add random delays to forwarded messages. In Stop-and-Go mixes, the delay that a mix adds to a traversing message follows an exponential distribution chosen by the sender [113]. To defeat $n-1$ attacks, the sender also provides per-hop time windows for a message, and each mix needs to check whether the arrival time of a message lies within the time window. Thus, an attacker that holds a message for a specific time to launch the $n-1$ attack will be detected. Dingledine, Serjantov, and Syverson further generalized Stop-and-Go mixes and proposed α mixing and τ mixing, which allow the sender to choose the desired trade-off between message delays and security [81].

Another solution to the blending attack is based on chaff traffic and anomaly detection. In the RGB-mix proposed by Danezis and Sassaman, a mix sends heartbeat chaff messages to itself through the mix network. The mix can detect the blending attack by observing the cutoff of the heartbeat messages. Upon detection, the mix will send additional chaff traffic to conceal the victim message until the attack stops [73].

7.1.4 Continuous Mixes

Due to the long end-to-end delays introduced by mixing strategies, traditional Mix networks are only suitable for delay-tolerant applications. Pfitzmann et al. first proposed to adopt the idea of the Mix networks to offer real-time calls within ISDN networks [145]. Later, the design evolved into a framework that can support real-time and low-latency anonymous communication [109]. Further research extended the techniques to offer anonymous web browsing [43].

To reduce latency, the above work proposes to separate the lifetime of an anonymous communication flow into a setup phase at the beginning and a data phase. Only the setup phase involves computation-heavy asymmetric cryptographic operations, and the data phase uses symmetric cryptographic operations, reducing processing latency.

Different from traditional Mix networks using batching or delays, the set of work lets each user transmit a fixed number of packets during each time slice, adding chaff packets if necessary. Furthermore, setup messages are mixed and synchronized among all senders to prevent traffic analysis against the setup phase.

TARANET targets the same problem of supporting high-volume, low-latency anonymous communication. Both this body of work and TARANET use end-to-end constant-rate transmission. However, this body of work fails to address the changing dynamics of flows in the presence of network jitters and packet drops, which become recognizable features that facilitate traffic analysis.

7.2 Onion Routing Systems

Onion routing systems aim to offer bidirectional low-latency anonymous communication. Conceptually, an onion routing system is a free-route Mix network without mixing. A client routes its traffic through a sequence of servers, called *onion routers*, to evade being traced by adversaries.

Similar to ISDN mixes [145], an onion routing system avoids expensive per-message asymmetric cryptographic operations. Instead, it groups messages into *circuits*. Asymmetric cryptographic operations only happen at the beginning of a circuit for creating shared keys between the sender and each involved router. An onion router applies only fast symmetric cryptographic operations to messages by reusing the shared keys.

An onion routing system guarantees anonymity by offering bitwise unlinkability for underlying messages and concealing circuit paths. An onion router that forwards a message knows only the previous onion router and the next one on the path. However, onion routing systems cannot resist either passive or

active traffic analysis [46,53,84,96,101,102,103,104,112,118,127,133,133,187,189,189,199,200], because these systems intentionally avoid delay-introducing mixing strategies or bandwidth-consuming chaff traffic.

7.2.1 Onion Routing Systems Before Tor

US Naval Research Lab (NRL) introduced the initial onion routing system design in 1996 [94]. The essential ideas for onion routing systems, i.e., symmetric cryptographic operations only for data messages, layered encryption/decryption for bitwise unlinkability, and hiding path information, are already present. The first-generation onion routing system also has design choices that are different from the current onion routing system [79]. First, rather than negotiating shared keys with onion routers, a sender distributes the symmetric keys to be used to the onion routers in the first message. Second, a sender can appoint an onion router to choose a path to the next specified onion router, and the sender does not know the symmetric keys for routers on the selected path.

The second-generation onion routing system explores the possibility of extending the first-generation onion routing system's defense to defeat traffic analysis [153]. It proposes to add link-level chaff to form constant-rate transmission between two onion routers. It also uses the real-time mixing on onion routers: an onion router reorders messages received within a fixed time interval but keeps the message order within the same circuit. While the new mechanisms protect data messages from passive traffic analysis, the resulting system still leaves the setup messages vulnerable to passive timing attacks [41]. Moreover, because both adding link-level chaff and mixing happen on onion routers, compromised onion routers can still launch active attacks by creating timing signatures or tagging messages.

The second-generation onion routing system also improves flexibility by allowing clients to join without forwarding traffic for others and enabling routers to serve only selected clients. Accordingly, the system adds entrance and exit policies that regulate which clients and traffic types an onion router is willing to support. With the separation of clients and routers and the increased flexibility, the second-generation onion routing system saw an unprecedented rise of deployment that in turn improves anonymity [29,78].

The Freedom Network [38,48] is the first commercial deployment of an onion routing system and the first one to witness large-scale deployment. Freedom features four differences from the second-generation onion routing system. First, Freedom uses the UDP transport instead of the TCP transport. Second, Freedom data encryption uses the block cipher as opposed to the stream cipher. Third, Freedom integrates a pseudonym and certificate service. Finally, Freedom pays third-party servers for forwarding traffic.

7.2.2 Tor

Since proposed in 2003, the third-generation onion routing design, Tor [79], has become the most popular anonymous communication system [25]. Tor anonymizes TCP streams by relaying the traffic through a path of three routers and is well engineered for proxying web traffic. Tor introduces the following main improvement compared to the second-generation onion routing designs:

- Tor drops the real-time mixing and the link-level traffic shaping because the proposed techniques resist neither passive nor active traffic analysis but incur additional latency and bandwidth overhead. Tor discards expensive yet ineffective traffic analysis resistance but pursues usability and flexibility to increase its user base to improve anonymity [39,78]. Specifically, the Tor project [26] introduces a variety of user-friendly applications. For example, the Tor Browser enables anonymous web browsing on multiple platforms [23]. The Tor Button sanitizes identity-leaking dynamic contents from the Firefox browser [24]. The Orbot proxy routes traffic from Android phones through the Tor network [16]. TorBirdy enables Tor for the Thunderbird email client [28]. Tails directly boot a Tor-enabled OS from a USB stick [22]. Moreover, Tor also has various pluggable transport [82, 87, 129, 190, 191, 192] to camouflage Tor traffic so that clients from censoring countries can connect to the Tor network.

- A centralized directory service maintains the topology and onion routers' certificates. A client can download the topology and independently decide which routers to use.

- A client chooses three onion routers to form a circuit rather than a random number of onion routers according to a binomial distribution in the second-generation onion routing system.

- Clients set up circuits in an iterative way [79] as opposed to the recursive way in the previous generations. A client first establishes a circuit to the first onion router and then always extends the circuit to the n-th onion router through the established circuit from the first onion router to the $(n-1)$-th. The iterative circuit setup allows the client to negotiate shared symmetric keys with each traversed onion router using the authenticated ephemeral Diffie-Hellman key exchange. As both the Diffie-Hellman key pairs and the negotiated symmetric keys are short-term ones, the process guarantees perfect forward secrecy [79] and compulsion resistance [68].

- Tor enables *hidden services* that allows servers to serve contents anonymously. To use the hidden service, a Tor client creates a circuit to an onion router specified by the hidden server called the *introduction point* to exchange handshake information. Then both the client and the hidden server establishes circuits to another onion router appointed by the client named *rendezvous point* to transmit data. Using the intermediaries, i.e., the introduction point and the rendezvous point, offers

anonymity for both senders and receivers. Splitting the functions of the handshake process and the data transmission between two onion routers allows the hidden service to continue ongoing connections through rendezvous points even when the censor blocks the introduction points.

7.2.3 Security Issues of Tor

Tor's anonymity guarantee is semi-trust based: given a Tor network of n onion routers with c of them compromised (or monitored), an attacker capable of performing traffic analysis succeeds with the probability of $\frac{c^2}{n^2}$ [86]. However, recent research demonstrated that the above model fails in the real world: attackers can succeed with high probability without compromising or physically present at a majority of onion routers.

- Because disjoint links within the overlay can intersect physically in the underlying Internet topology, a single ISP, an AS, or an IXP can observe traffic of both the entry and exit routers and launch traffic analysis attacks [40, 83, 85, 111, 134], breaking the anonymity guarantee. To fix the issue, a client should consider the underlying Internet topology information when picking a circuit's path [30, 40, 83, 111]. Furthermore, a compromised AS can use routing attacks to increase the probability that it observes both entry and the exit routers [170]. Therefore, it is also necessary to monitor the activity of the Internet [169].

- To balance the load among onion routers, a client selects its circuit's path considering available bandwidth of onion routers [27]. Recent research also considers latency [37, 139, 140] and traffic congestion [186] during the path selection to improve the performance. However, Wacek et al. showed that there is a trade-off between anonymity and performance when selecting paths [184]. Focusing on circuits' performance inevitably reduces the diversity of the selected onion routers, again weakening the anonymity guarantee. Furthermore, an adversarial router can intentionally misreport its bandwidth to increase the probability to be chosen. Thus, additional secure measurement are needed [110, 143, 166, 167].

- An attacker can use the Denial-of-Service attacks to slow down benign onion routers causing observable effects (e.g., increased delays) on victim circuits [84, 133]. An adversary can also exploit the Tor protocol to shut down benign onion routers to increase the probability to be chosen [108]. Overlier and Syverson demonstrated that an adversary can use the attack for de-anonymizing hidden services [138].

7.2.4 Improving Tor Performance

One primary body of research aims to improve Tor's performance. There are two main lines of research focusing on 1) circuit path selection and 2) Tor's congestion control respectively, which are among the six problems of Tor performance identified by Dingledine and Murdoch [80].

The path selection algorithm can significantly influence the communication quality of the resulting circuits. Tor's path selection weighs onion routers proportionally regarding their bandwidth to balance the load among onion routers [27, 79]. To prevent adversarial routers from falsely reporting large bandwidth to attract more traffic [41], the current Tor design proposes to employ monitoring servers called measurement authorities to scan routers' bandwidth [143, 166]. However, adversarial routers can still identify the measurement adversaries and misreport [174]. Eigenspeed [167] and Peerflow [110] enable each router to measure the connections to other routers and report to a measurement authority that calculates a synthesized bandwidth measurement by integrating all reports.

Besides available bandwidth, researchers also propose to consider latency and network congestion for path selection. Sherr et al. [161, 162] suggested mapping onion routers onto a virtual coordinate system [65], where the Euclidean distance between a pair of routers indicates the latency of a connection between them. Thus, finding a circuit with the shortest latency is equivalent to discovering a short path in the coordinate system. Similarly, LASTor [30] uses the geographical distance in place of the virtual coordinate as an estimate for latency. Wang et al. [186] proposed to measure the Round-Trip Times (RTT) during circuit construction and to select circuits with the minimum RTTs to avoid congested routers.

Another source of Tor's suboptimal performance is Tor's congestion control [80]. Because all circuits on the same link share the TCP connection, inter-circuit interference and fair circuit-level scheduling become the performance bottlenecks. Additionally, each end-to-end TCP stream uses a fixed-size sliding window for flow control, which lacks adaptivity when interacting with the dynamics of per-hop TCPs, causing excessive queuing in both circuit-level and socket-level queues [179].

Regarding the inter-circuit interference, because of the TCP congestion control, one packet drop of a circuit stalls all other circuits on the same connection, which is called the head-of-line blocking [80]. Tor with TCP-over-DTLS [152] and PCTCP [33] suggest using per-circuit transport between a pair of routers, and IMUX [91] and Torchestra [97] propose to multiplex circuits on a link over multiple TCP connections. Nowlan et al. [136] advocated out-of-order delivery for the per-hop TCP.

Regarding the fair scheduling, the traditional Tor circuit scheduling algorithm lacks inter-circuit fairness [177, 178]. Tang and Goldberg proposed to let each onion router to monitor the Exponentially Weighted Moving Average (EWMA) metric of each flow and schedule circuits accordingly [173]. Jan-

son, Syverson, and Hopper proposed three adaptive throttling algorithms with routers' local knowledge to prevent a small number of users to consume a majority of the bandwidth [107]. DiffTor uses machine learning algorithms to categorize the circuits into interactive ones and bulk downloading ones to prioritize the interactive traffic more effectively [31].

Regarding the excessive queuing delays because of the inefficient end-to-end transport control, Jansen et al. [106] identified the socket-level queuing as one missed source of the end-to-end latency and proposed KIST that considers kernel information when scheduling circuits to avoid delays. DefenstraTor [32] replaces the end-to-end sliding window of the TCP stream with a hop-by-hop window, which is a design adapted from ATM networks. UDP-OR [183] suggests a clean-slate design to use an end-to-end TCP through the entire circuit without per-hop TCPs. BackTap [179] proposes to combine latency-based congestion control for per-hop TCP and the backpressure-based flow control for inter-hop signaling to improve performance and increase fairness.

7.3 Peer-to-peer Anonymous Communication Systems

Both Chaum's Mix network and the first-generation Tor require each participant to forward traffic for others to improve anonymity. Peer-to-peer anonymous communication systems extend the same assumption to a large peer-to-peer network with transient nodes for better scalability. Depending on whether the path selection depends on a structured Distributed Hash Table (DHT), we classify the body of work into unstructured and structured peer-to-peer anonymous communication systems.

7.3.1 Unstructured Peer-to-peer Anonymous Communication Systems

Tarzan [88] is a peer-to-peer anonymous communication system that aims to defeat traffic analysis attacks. Tarzan's communication model and packet formats are similar to Tor. Tunnel creation and data transmission are separated to offer symmetric-cryptography-only packet forwarding for data messages. Tarzan also uses layered encryption for bitwise unlinkability and conceals path information in headers. To resist traffic analysis, each node (called a *mimic*) maintains long-term connections with neighbor nodes. A node exchanges chaff traffic with neighbor nodes to hide the existence and the metadata of data messages.

Because of the network's dynamic nature, the original Tarzan design only requires each node to know a subset of other nodes for scalability. However, Clayton and Danezis [67] showed that in a large network where each node only knows a small subset of other nodes, a single adversary node by knowing its previous and next nodes on the path can identify the sender with high probability. To defeat this attack,

the final version of Tarzan requires each node to know all other nodes within the network through a gossip protocol, sacrificing scalability for anonymity.

MorphMix [155] shares the same architecture and threat model with Tarzan. The primary difference lies in the path selection. MorphMix proposes to let the intermediate node, instead of the sender, to determine the route to forward messages. A *route capture* attack is possible: a malicious node may only route messages through other colluding nodes. MorphMix proposes a collusion detection mechanism that unveils any cliques during node selection. However, Tabriz and Borisov demonstrated that an attack to circumvent the detection of MorphMix's collusion detection mechanism [172].

7.3.2 Structured Peer-to-peer Anonymous Communication Systems

Another body of work uses a structured approach. The structured peer-to-peer anonymous communication systems use Distributed Hash Tables (DHT) to manage and distribute network topology information for higher scalability. Salsa [135] uses a customized secure lookup operation to select random relays in the network. AP3 [124] performs a stochastic expected length random walk combined with DHT lookup to determine relays on a path. Cashmere [201] forms groups of relays near a given ID within a DHT, and any relay from a group can forward a message to the next relay. However, the above systems are vulnerable to the information leakage attacks [126] and the selective Denial-of-Service attacks [47, 176].

NISAN [141] uses redundant DHT lookups to mitigate information leakage attacks, and hides the destination by downloading the entire routing table from intermediate nodes. Torsk [123] enables senders to use secret buddy nodes to perform DHT lookup to prevent information leakage attacks. Nevertheless, Wang et al. showed that both of the systems are still vulnerable to passive and active attacks [185]. ShadowWalker [125] uses random walks in a redundant DHT to form paths, and each sender employs multiple *shadows* to check and sign routing tables on other nodes. Unfortunately, Schuchard et al. found that ShadowWalker is vulnerable to the *eclipse attack* [158] where a node with compromised shadows infects routing tables of others.

7.4 Private Messaging Systems

Recent research focuses on anonymous communication systems that provably defeat traffic analysis attacks as long as a benign server exists. Though the underlying mechanisms can vary, these systems share similar characteristics: strong anonymity but poor performance and scalability. Thus, such systems are suitable for applications such as secure messaging and anonymous micro-blogging.

One category of private messaging systems is based on the Dining Cryptographer (DC) problem. Chaum first investigated the DC problem about how to enable a set of communication entities to have information-theoretic anonymity and designed Dining-Cryptographer network (DC-net) accordingly [54]. The initial DC-net suffers from practical problems: a single malicious participant can disrupt the protocol without detection. Golle and Juels [95] proposed asymmetric techniques that make DC-net robust against disruption. Dissent further offers one-on-one correspondence between messages and participants to defeat Denial-of-Service attacks and Sybil attacks [63].

To scale the DC-net and reduce both bandwidth and computation overhead, Herbivore [93] uses a trusted leader to mediate the broadcast channel to reduce bandwidth overhead. Dissent [63, 193] and Verdict [64] constructs DC-nets through use of verifiable shuffle algorithms and bulk transfer protocol. Both systems extend DC-nets to use a set of decentralized servers for higher scalability. However, the state-of-the-art DC-nets still broadcast all messages and require computation overhead quadratic to the number of messages sent.

Another category of private messaging systems uses the Private Information Retrieval (PIR) technique. In Riffle [114], clients post messages to a small number of central servers and pull messages of interest. Riffle uses PIR for receiver anonymity and uses the verifiable shuffle for sender anonymity. However, Riffle incurs high client load and cannot handle network churn well. A member's join or leave results in redoing the setup phase for all sessions. In Pynchon Gate [157], clients use a Mix network to upload messages to email servers and receive messages anonymously through PIR. Riposte [62] proposes a reverse PIR technique to anonymize posted messages and reduces clients' bandwidth cost within broadcast channels. Pung [36] organizes message storage using a key-value store and leverages PIR for anonymous message posting and retrieval. Pung also enables users to retrieve multiple messages through a single request through a muli-retrieval scheme. Using PIR schemes incur high computation overhead and limits the scalability. Moreover, these PIR-based systems incur high latency because of polling and high computation overhead.

Another line of work tackles the anonymity problem in a differential privacy manner. Vuvuzela [181] combines a set of previously proposed techniques, the cascade mixing, end-to-end constant-rate transmission, and end-to-end chaff traffic but carefully engineers the system to minimize the number of variables observed by an attacker. Vuvuzela also uses the differential privacy technique to add noise to the exposed variables to provably hide information about underlying communication. However, the use of the cascaded mixing servers poses a fundamental challenge for such a system to scale. Stadium [180] aims to improve the scalability by replacing Vuvuzela's cascade mixing with verifiable parallel mixing. Stadium additionally proposes distributed noise generation and differentially-private routing to offer provable dif-

ferential privacy in the context of the parallel mixing. Nevertheless, compared to Vuvuzela, Stadium requires more noise and computation, which yield significant longer latency up to hundreds of seconds.

Chapter 8

Conclusion and Future Work

Anonymity as a fundamental right faces increasing challenges from both profit-driven private companies and government surveillance programs. However, current overlay-based anonymous communication solutions at the application layer fail to offer strong anonymity and high performance. As the research on Future Internet Architectures continues to emerge, this dissertation explores the viability and benefits of redesigning anonymous communication as a core service of the network infrastructure that supports cryptographic operations on routers and path control. Our results demonstrate that infrastructure-based anonymous communication systems can achieve 1-2 orders of magnitude higher throughput than the state-of-the-art overlay-based solutions and simultaneously offer anonymity in the presence of strong deanonymization attacks.

We also show a clear trade-off between performance/scalability and the level of guaranteed anonymity that an infrastructure-based anonymity system can offer. Specifically, we design three anonymity systems, i.e., PHI, HORNET, and TARANET, targeting a wide range of adversaries from a single malicious ISP through a global passive adversary incapable of traffic analysis attacks to a global active adversary with different efficiency and scalability.

This dissertation aims to delineate a new boundary of performance, scalability, and security of future anonymity systems, reveal potential issues and difficulties, and develop new techniques that can guide or be integrated into both existing and future anonymity protocols. For example, HORNET proposes an onion routing packet format with the packet-carried cryptographic state. PHI presents a highly efficient packet header format that hides path information. TARANET advocates end-to-end constant-rate transmission that fights traffic analysis attacks and accommodates network jitters and packet drops by packet splitting.

We remark that this work provides neither a complete ready-to-deploy solution of an infrastructure-

based anonymity system nor a complete enumeration of all possible designs. Looking forward, we would continue to tackle practical issues of implementing and deploying our solutions and to explore new designs that can offer stronger anonymity or higher performance. We conclude the dissertation by listing future work as follows.

- *Anonymous topology information service*. One key missing component of our infrastructure-based anonymity systems is the topology information service that distributes the topology information to end hosts. While there exist multiple solutions disseminating topology information within an overlay, it remains an open research question how to design such systems in an anonymous, efficient, and scalable manner in the context of the emerging FIAs.

- *Comparing anonymity between overlay-based and infrastructure-based designs*. On the one hand, because overlay-based anonymity system is unaware of underlying network topology, a compromised ISP can intercept multiple links within the overlay network. On the other hand, lacking global redirection, infrastructure-based anonymity systems are subject to inherent path information leakage. Therefore, it remains an open question which system has higher anonymity given a set of compromised ISPs.

- *Infrastructure-based anonymity systems assisted by trusted computing technology*. Recent advances in the trusted computing technology have enabled high-performance execution of attested code, which in turn leads to realistic secure multi-party computation. Secure multi-party computation is an essential building block of a series of anonymity protocol with provable anonymity under the anytrust assumption. Another line of future work is to explore the possibility of using the trusted computing technology to build efficient and scalable infrastructure-based anonymity systems to achieve stronger and provable anonymity guarantees.

Bibliography

[1] The CAIDA anonymized Internet traces 2015. http://www.caida.org/data/passive/passive_2015_dataset.xml. Retrieved on April 11, 2016.

[2] CAIDA AS-relationship dataset. http://www.caida.org/data/as-relationships/. Retrieved on April 28, 2016.

[3] CAIDA UCSD anonymized Internet traces 2014. http://www.caida.org/data/passive/passive_2014_dataset.xml. Retrieved on April 30, 2015.

[4] Cambridge analytica used data from facebook and politico to help trump. "https://www.theguardian.com/technology/2017/oct/26/cambridge-analytica-used-data-from-facebook-and-politico-to-help-trump". Retrieved on April 4, 2018.

[5] Cisco ASR-1000. http://www.cisco.com/c/en/us/products/routers/. Retrieved on April 28, 2015.

[6] Cisco routers. http://www.cisco.com/c/en/us/products/routers. Retrieved on August 5, 2015.

[7] Curve25519-donna. https://code.google.com/p/curve25519-donna/. Retrieved on December 13, 2014.

[8] DPDK: Data plane development kit. http://dpdk.org/. Retrieved on December 23, 2014.

[9] How the NSA's domestic spying program works. "https://www.eff.org/nsa-spying/how-it-works". Retrieved on November 3, 2015.

[10] Intel AESNI sample library. https://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library. Retrieved on December 13, 2014.

[11] The invisible Internet project. https://geti2p.net/en/. Retrieved on February 23, 2016.

[12] iPlane dataset. http://iplane.cs.washington.edu/data/data.html. Traceroute data was generated on October 12, 2014.

[13] The JonDo project. "https://anonymous-proxy-servers.net/". Retrieved on March 27, 2018.

[14] JonDonym anonymous proxy servers. https://anon.inf.tu-dresden.de/index_en.html. Retrieved on February 23, 2016.

[15] NSA targets the privacy-conscious. http://daserste.ndr.de/panorama/aktuell/NSA-targets-the-privacy-conscious,nsa230.html. Retrieved on 2015.05.13.

[16] Orbot: Tor for android. https://guardianproject.info/apps/orbot/. Retrieved on April 11, 2018.

[17] PolarSSL. https://polarssl.org/. Retrieved on December 13, 2014.

[18] RouteView project. http://www.routeviews.org/. Retrieved on December 13, 2014.

[19] Segment routing architecture (IETF draft). https://datatracker.ietf.org/doc/draft-ietf-spring-segment-routing/. Retrieved on May 13, 2015.

[20] Spirent TestCenter. http://www.spirent.com/Ethernet_Testing/Software/TestCenter. Retrieved on December 23, 2014.

[21] Surveillance techniques: how your data become our data. "https://nsa.gov1.info/surveillance/". Retrieved on November 3, 2015.

[22] Tails: the amnesic incognito live system. https://tails.boum.org/. Retrieved on April 11, 2018.

[23] The Tor browser. https://www.torproject.org/projects/torbrowser.html.en. Retrieved on April 11, 2018.

[24] Tor button. https://www.torproject.org/docs/torbutton/. Retrieved on April 11, 2018.

[25] Tor metrics: Direct users by country. "https://metrics.torproject.org/userstats-relay-country.html. Retrieved on November 3, 2015.

[26] Tor project. https://www.torproject.org/. Retrieved on February 23, 2016.

[27] Tor specification. https://github.com/torproject/torspec. Retrieved on April 11, 2018.

[28] Torbirdy. https://trac.torproject.org/projects/tor/wiki/torbirdy. Retrieved on April 11, 2018.

[29] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. In *Proceedings of the International Conference on Financial Cryptography (FC)*, 2003.

[30] Masoud Akhoondi, Curtis Yu, and Harsha V Madhyastha. Lastor: A low-latency AS-aware Tor client. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2012.

[31] Mashael AlSabah, Kevin Bauer, and Ian Goldberg. Enhancing Tor's performance using real-time traffic classification. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[32] Mashael AlSabah, Kevin Bauer, Ian Goldberg, Dirk Grunwald, Damon McCoy, Stefan Savage, and Geoffrey M Voelker. Defenestrator: Throwing out windows in Tor. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2011.

[33] Mashael AlSabah and Ian Goldberg. PCTCP: per-circuit TCP-over-IPsec transport for anonymous communication overlay networks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2013.

[34] Ashok Anand, Fahad Dogar, Dongsu Han, Boyan Li, Hyeontaek Lim, Michel Machado, Wenfei Wu, Aditya Akella, David G Andersen, John W Byers, et al. XIA: An architecture for an evolvable and trustworthy internet. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2011.

[35] Tom Anderson, Ken Birman, Robert Broberg, Matthew Caesar, Douglas Comer, Chase Cotton, Michael J Freedman, Andreas Haeberlen, Zachary G Ives, Arvind Krishnamurthy, et al. The NEBULA future Internet architecture. *The Future Internet*, pages 16–26, 2013.

[36] Sebastian Angel and Srinath TV Setty. Unobservable communication over fully untrusted infrastructure. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[37] Robert Annessi and Martin Schmiedecker. Navigator: Finding faster paths to anonymity. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.

[38] Adam Back, Ian Goldberg, and Adam Shostack. Freedom 2.1 security issues and analysis. *White Paper, Zero Knowledge Systems, Inc.*, 2001.

[39] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proceedings of the International Workshop on Information Hiding (IH)*, 2001.

[40] Armon Barton and Matthew Wright. DeNASA: Destination-naive AS-awareness in anonymous communications. *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[41] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against Tor. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2007.

[42] Daniel J Bernstein. Curve25519: new Diffie-Hellman speed records. In *Proceedings of the International Conference on Theory and Practice in Public-Key Cryptography (PKC)*, 2006.

[43] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable internet access. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2001.

[44] Oliver Berthold and Heinrich Langos. Dummy traffic against long term intersection attacks. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2003.

[45] Violet Blue. Facebook turns user tracking 'bug' into data mining 'feature' for advertisers. `http://www.zdnet.com/article/facebook-turns-user-tracking-bug-into-data-mining-feature-for-advertisers/`. Retrieved on April 11, 2018.

[46] Avrim Blum, Dawn Song, and Shobha Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[47] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[48] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture, 2001. White paper, Zero Knowledge Systems, Inc.

[49] Zach Brown. Cebolla: Pragmatic IP anonymity. In *Proceedings of the Ottawa Linux Symposium*, 2002.

[50] R. Bush and R. Austein. The resource public key infrastructure (RPKI) to router protocol. IETF RFC 6810.

[51] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In *Proceedings of the IACR International Cryptology Conference (CRYPTO)*, 2005.

[52] Sambuddho Chakravarty, Marco V. Barbera, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. On the effectiveness of traffic analysis against anonymity networks using flow records. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, 2014.

[53] Sambuddho Chakravarty, Angelos Stavrou, and Angelos D Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2010.

[54] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[55] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[56] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[57] Chen Chen, Daniele E. Asoni, David Barrera, George Danezis, and Adrian Perrig. HORNET: High-speed onion routing at the network layer. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2015.

[58] Chen Chen, Daniele E. Asoni, Adrian Perrig, David Barrera, George Danezis, and Carmela Troncoso. TARANET: Traffic-analysis resistant anonymity at the network layer. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[59] Chen Chen, Daniele Enrico Asoni, David Barrera, George Danezis, and Adrian Perrig. HORNET: High-speed Onion Routing at the Network Layer. *arXiv/1507.05724*, July 2015.

[60] Chen Chen and Adrian Perrig. PHI: Path-hidden lightweight anonymity protocol at network layer. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2017.

[61] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6), 1998.

[62] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2015.

[63] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010.

[64] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *Proceedings of the USENIX Security Symposium*, 2013.

[65] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review*, 34(4):15–26, 2004.

[66] George Danezis. The traffic analysis of continuous-time Mixes. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2004.

[67] George Danezis and Richard Clayton. Route fingerprinting in anonymous communications. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2006.

[68] George Danezis and Jolyon Clulow. Compulsion resistant anonymous communications. In *Proceedings of the International Workshop on Information Hiding (IH)*, 2005.

[69] George Danezis, Claudia Diaz, Carmela Troncoso, and Ben Laurie. Drac: An architecture for anonymous low-volume communications. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2010.

[70] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2003.

[71] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure Mix format. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2009.

[72] George Danezis and Ben Laurie. Minx: A simple and efficient anonymous packet format. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2004.

[73] George Danezis and Len Sassaman. Heartbeat traffic to counter (n-1) attacks: red-green-black mixes. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2003.

[74] George Danezis and Andrei Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *Proceedings of the International Workshop on Information Hiding (IH)*, 2005.

[75] Drew Dean and Adam Stubblefield. Using client puzzles to protect TLS. In *Proceedings of the USENIX Security Symposium*, 2001.

[76] Claudia Diaz and Bart Preneel. Reasoning about the anonymity provided by pool mixes that generate dummy traffic. In *Proceedings of the International Workshop on Information Hiding (IH)*, 2004.

[77] Claudia Diaz and Andrei Serjantov. Generalising Mixes. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2003.

[78] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In *Proceedings of the Workshop on the Economics of Information Security (WEIS)*, 2006.

[79] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the USENIX Security Symposium*, 2004.

[80] Roger Dingledine and Steven J. Murdoch. Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. "`http://www.torproject.org/press/presskit/2009-03-11-performance.pdf`, 2009. Retrieved on May 23, 2016.

[81] Roger Dingledine, Andrei Serjantov, and Paul Syverson. Blending different latency traffic with alpha-mixing. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2006.

[82] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2013.

[83] Matthew Edman and Paul Syverson. AS-awareness in Tor path selection. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.

[84] Nathan S Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on Tor using long paths. In *Proceedings of the USENIX Security Symposium*, 2009.

[85] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2004.

[86] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Probabilistic analysis of onion routing in a black-box model. *ACM Transactions on Information and System Security (TISSEC)*, 15(3):14:1–14:28, November 2012.

[87] David Fifield. Meek: A simple HTTP transport and big ideas. `https://lists.torproject.org/pipermail/tor-dev/2014-January/006159.html`, 2014. Retrieved on April 11, 2018.

[88] Michael J Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2002.

[89] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2004.

[90] Sean Gallagher. Facebook scraped call/text/message data for years from android phones. "`https://arstechnica.com/information-technology/2018/03/facebook-scraped-call-text-message-data-for-years-from-android-phones/?amp=1`". Retrieved on April 4, 2018.

[91] John Geddes, Rob Jansen, and Nicholas Hopper. IMUX: Managing tor connections from two to infinity, and beyond. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2014.

[92] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2009.

[93] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[94] David M Goldschlag, Michael G Reed, and Paul F Syverson. Hiding routing information. In *Proceedings of the International Workshop on Information Hiding (IH)*, 1996.

[95] Philippe Golle and Ari Juels. Dining cryptographers revisited. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2004.

[96] Xun Gong, Nikita Borisov, Negar Kiyavash, and Nabil Schear. Website detection using remote traffic analysis. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2012.

[97] Deepika Gopal and Nadia Heninger. Torchestra: Reducing interactive traffic delays over tor. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2012.

[98] Ceki Gülcü and Gene Tsudik. Mixing E-mail with Babel. In *IEEE NDSS*, 1996.

[99] Ceki Gülcü and Gene Tsudik. Mixing email with Babel. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 1996.

[100] Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, and Bradley Huffaker. On routing asymmetry in the Internet. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2005.

[101] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2), 2010.

[102] Amir Houmansadr and Nikita Borisov. SWIRL: A Scalable Watermark to Detect Correlated Network Flows. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2011.

[103] Amir Houmansadr and Nikita Borisov. The need for flow fingerprints to link correlated network flows. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2013.

[104] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. RAINBOW: A robust and invisible non-blind watermark for network flows. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2009.

[105] Hsu Chun Hsiao, Tiffany Hyun Jin Kim, Adrian Perrig, Akira Yamada, Samuel C. Nelson, Marco Gruteser, and Wei Meng. LAP: Lightweight anonymity and privacy. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2012.

[106] Rob Jansen, John Geddes, Chris Wacek, Micah Sherr, and Paul F. Syverson. Never been KIST: Tor's congestion management blossoms with kernel-informed socket transport. In *Proceedings of the USENIX Security Symposium*, 2014.

[107] Rob Jansen, Paul F Syverson, and Nicholas Hopper. Throttling tor bandwidth parasites. In *Proceedings of the USENIX Security Symposium*, 2012.

[108] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the tor network. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.

[109] Anja Jerichow, Jan Müller, Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. Real-Time MIXes: A Bandwidth-Efficient Anonymity Protocol. *IEEE Journal on Selected Areas in Communications*, 16(4), 1998.

[110] Aaron Johnson, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson. Peerflow: Secure load balancing in tor. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2017.

[111] Aaron Johnson, Rob Jansen, Aaron D Jaggard, Joan Feigenbaum, and Paul Syverson. Avoiding the man on the wire: Improving Tor's security with trust-aware path selection. In *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2017.

[112] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[113] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *Proceedings of the International Workshop on Information Hiding (IH)*, 1998.

[114] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle. *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[115] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[116] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. Towards efficient traffic-analysis resistant anonymity networks. In *ACM SIGCOMM*, 2013.

[117] Taeho Lee, Christos Pappas, Adrian Perrig, Virgil Gligor, and Yih-Chun Hu. The case for in-network replay suppression. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2017.

[118] Brian N Levine, Michael K Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix-based systems. In *Proceedings of the International Conference on Financial Cryptography (FC)*, 2004.

[119] Shannon Liao. Google admits it tracked user location data even when the setting was turned off. `https://www.theverge.com/2017/11/21/16684818/google-location-tracking-cell-tower-data-android-os-firebase-privacy`. Retrieved on April 11, 2018.

[120] Vincent Liu, Seungyeop Han, Arvind Krishnamurthy, and Thomas Anderson. Tor instead of IP. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2011.

[121] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, K. Claffy, and A. Vahdat. The Internet AS-level topology: Three data sources and one definitive metric. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2006.

[122] Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *PETS*, 2005.

[123] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with torsk. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.

[124] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S Wallach. Ap3: Cooperative, decentralized anonymous communication. In *Proceedings of the ACM SIGOPS European Workshop*, 2004.

[125] Prateek Mittal and Nikita Borisov. Shadowwalker: peer-to-peer anonymous communication using redundant structured topologies. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.

[126] Prateek Mittal and Nikita Borisov. Information leaks in structured peer-to-peer anonymous communication systems. *ACM Transactions on Information and System Security (TISSEC)*, 15(1):5, 2012.

[127] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011.

[128] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *USENIX Security*, 2011.

[129] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[130] Bodo Möller. Provably secure public-key encryption for length-preserving chaumian mixes. In *Proceedings of the RSA conference on The cryptographers' track*, 2003.

[131] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol v. 2. IETF Draft, 2003.

[132] R. Moskowitz and P. Nikander. Host identity protocol (HIP) architecture. IETF RFC 4423.

[133] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2005.

[134] Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by Internet-Exchange-level adversaries. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2007.

[135] Arjun Nambiar and Matthew Wright. Salsa: a structured approach to large-scale anonymity. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2006.

[136] Michael F Nowlan, David Isaac Wolinsky, and Bryan Ford. Reducing latency in Tor circuits with unordered delivery. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2013.

[137] Luke O'Connor. On blending attacks for mixes with memory. In *Proceedings of the International Workshop on Information Hiding (IH)*, 2005.

[138] Lasse Overlier and Paul Syverson. Locating hidden servers. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2006.

[139] Andriy Panchenko, Fabian Lanze, and Thomas Engel. Improving performance and anonymity in the Tor network. In *Proceedings of the IEEE Performance Computing and Communications Conference (IPCCC)*, 2012.

[140] Andriy Panchenko and Johannes Renner. Path selection metrics for performance-improved onion routing. In *Proceedings of the IEEE International Symposium on Applications and the Internet (SAINT)*, 2009.

[141] Andriy Panchenko, Stefan Richter, and Arne Rache. NISAN: network information service for anonymization networks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.

[142] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. *SCION: A Secure Internet Architecture*. Springer International Publishing AG, 2017.

[143] Mike Perry. Torflow: Tor network analysis. In *Proceedings of the Workshop of Hot Topics in Privacy Enhancing Technologies (HotPETS)*, 2009.

[144] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity - a proposal for terminology. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2001.

[145] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. ISDN-Mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems (KiVS)*, 1991.

[146] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct RSA-implementation of mixes. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 1989.

[147] Ania Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In *Proceedings of the USENIX Security Symposium*, 2017.

[148] Ryan Pries, Wei Yu, Xinwen Fu, and Wei Zhao. A new replay attack against anonymous communication networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, 2008.

[149] Felix Putze, Peter Sanders, and Johannes Singler. Cache-, hash- and space-efficient bloom filters. In *Proceedings of the International Symposium on Experimental Algorithms (SEA)*, 2007.

[150] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(3):2–13, 2012.

[151] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2001.

[152] Joel Reardon and Ian Goldberg. Improving Tor using a TCP-over-DTLS tunnel. In *Proceedings of the USENIX Security Symposium*, 2009.

[153] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.

[154] Yakov Rekhter and Tony Li. A border gateway protocol 4 (BGP-4), 1995.

[155] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2002.

[156] Jody Sankey and Matthew Wright. Dovetail: Stronger anonymity in next-generation internet routing. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[157] Len Sassaman, Bram Cohen, and Nick Mathewson. The pynchon gate: A secure method of pseudonymous mail retrieval. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2005.

[158] Max Schuchard, Alexander W Dean, Victor Heorhiadi, Nicholas Hopper, and Yongdae Kim. Balancing the shadows. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2010.

[159] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In *Proceedings of the International Workshop on Information Hiding (IH)*, 2002.

[160] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2003.

[161] Micah Sherr, Matt Blaze, and Boon Thau Loo. Scalable link-based relay selection for anonymous routing. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2009.

[162] Micah Sherr, Andrew Mao, William R Marczak, Wenchao Zhou, Boon Thau Loo, and Matthew A Blaze. A^3: An extensible platform for application-aware anonymity. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2010.

[163] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P^5: A protocol for scalable anonymous communication. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2002.

[164] Erik Shimshock, Matt Staats, and Nick Hopper. Breaking and provably fixing minx. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[165] Matthew Smart, G. Robert Malan, and Farnam Jahanian. Defeating TCP/IP stack fingerprinting. In *Proceedings of the USENIX Security Symposium*, 2000.

[166] Robin Snader and Nikita Borisov. A tune-up for Tor: Improving security and performance in the Tor network. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.

[167] Robin Snader and Nikita Borisov. Eigenspeed: secure peer-to-peer bandwidth evaluation. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[168] Nova Spivack. The post-privacy world. `https://www.wired.com/insights/2013/07/the-post-privacy-world/`. Retrieved on April 11, 2018.

[169] Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. Counter-RAPTOR: Safeguarding Tor against active routing attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2017.

[170] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: routing attacks on privacy in Tor. In *Proceedings of the USENIX Security Symposium*, 2015.

[171] Srikanth Sundaresan, Walter de Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband Internet performance: a view from the gateway. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2011.

[172] Parisa Tabriz and Nikita Borisov. Breaking the collusion detection mechanism of MorphMix. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2006.

[173] Can Tang and Ian Goldberg. An improved algorithm for Tor circuit scheduling. In *Proceedings of the ACM Conference on Computer and Communication Security (CCS)*, 2010.

[174] Fabrice Thill. *Hidden Service Tracking Detection and Bandwidth Cheating in Tor Anonymity Network*. PhD thesis, Univ. Luxembourg, 2014.

[175] Craig Timberg. Verizon, AT&T tracking their users with 'supercookies'. `https://www.washingtonpost.com/business/technology/verizon-atandt-tracking-their-users-with-super-cookies/2014/11/03/7bbbf382-6395-11e4-bb14-4cfea1e742d5_story.html?utm_term=.6032bfa76ab8`. Retrieved on April 4, 2015.

[176] Andrew Tran, Nicholas Hopper, and Yongdae Kim. Hashing it out in public: common failure modes of dht-based anonymity schemes. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2009.

[177] Florian Tschorsch and Björn Scheuermann. Tor is unfair and what to do about it. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*, 2011.

[178] Florian Tschorsch and Björn Scheuermann. How (not) to build a transport layer for anonymity overlays. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):101–106, 2013.

[179] Florian Tschorsch and Björn Scheuermann. Mind the gap: Towards a backpressure-based transport protocol for the Tor network. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[180] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[181] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2015.

[182] Laurent Vanbever, Oscar Li, Jennifer Rexford, and Prateek Mittal. Anonymity on quicksand: Using bgp to compromise tor. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2014.

[183] Camilo Viecco. UDP-OR: A fair onion transport design. In *Proceedings of the Workshop of Hot Topics in Privacy Enhancing Technologies (HotPETS)*, 2008.

[184] Chris Wacek, Henry Tan, Kevin S Bauer, and Micah Sherr. An empirical evaluation of relay selection in Tor. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2013.

[185] Qiyan Wang, Prateek Mittal, and Nikita Borisov. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2010.

[186] Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware path selection for Tor. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2012.

[187] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the USENIX Security Symposium*, 2014.

[188] Wei Wang, Mehul Motani, and Vikram Srinivasan. Dependent link padding algorithms for low latency anonymity systems. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2008.

[189] Xinyuan Wang and Douglas S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2003.

[190] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: a camouflage proxy for the Tor anonymity system. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[191] Brandon Wiley. Dust: A censorship-resistant internet transport protocol. `https://github.com/blanu/Dust`. Retrieved on April 11, 2018.

[192] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, 2013.

[193] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[194] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. Hang with your buddies to resist intersection attacks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2013.

[195] Xiaowei Yang, David Clark, and Arthur W Berger. Nira: a new inter-domain routing architecture. *IEEE/ACM Transactions on Networking (ToN)*, 15(4):775–788, 2007.

[196] Bassam Zantout and Ramzi Haraty. I2P data communication system. In *Proceedings of the International Conference on Networking (ICN)*, 2011.

[197] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Kimberley Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM)*, 2014.

[198] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2011.

[199] Yin Zhang and Vern Paxson. Detecting stepping stones. In *Proceedings of the USENIX Security Symposium*, 2000.

[200] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. On flow correlation attacks and countermeasures in mix networks. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2004.

[201] Li Zhuang, Feng Zhou, Ben Y Zhao, and Antony Rowstron. Cashmere: Resilient anonymous routing. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.