© 2015 Zhuotao Liu

FLOWPOLICE: ENFORCING CONGESTION ACCOUNTABILITY TO DEFEND AGAINST DDOS ATTACKS

BY

ZHUOTAO LIU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Associate Professor Yih-Chun Hu

ABSTRACT

Defending the Internet against distributed denial of service (DDoS) attacks is a fundamental problem. Despite over a decade of research, little progress has been made on the real-world deployment of proposed approaches due to the prohibitive deployment hurdles. This thesis presents FlowPolice, a new DDoS defense mechanism capable of thwarting millions of attack flows, while requiring very lightweight deployment. Specifically, FlowPolice can immediately benefit the first deployed autonomous system (AS) without further deployment at other ASs, and a single deployed router can protect all downstream links that implement a simple prioritization mechanism. The design of FlowPolice suppresses attack traffic by forcing attackers to be accountable for congestion via proper rate limiting. To learn users' congestion accountability, FlowPolice leverages a capability feedback mechanism so that the deploying router can make rate limiting decisions based only on its self-generated capability tags.

We use theoretical analysis, large scale simulation and Linux implementation to demonstrate the effectiveness of FlowPolice. Specifically, the theoretical analysis proves that FlowPolice ensures per-flow fair share at the bottleneck link. Our implementation shows that FlowPolice can scale up to handle very large scale DDoS attacks and introduces little packet processing overhead. We also perform detailed packet-level simulation to show that FlowPolice is effective to mitigate DDoS attacks.

To my parents, friends, and colleagues for their love and support.

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. Yih-Chun Hu, for the advice and support he has given me along the way.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 DESIGN RATIONALES 4
2.1 Problem Space 4
2.2 Design Goals 4
2.3 Enforce Congestion Accountability 5
2.4 Fairness Guarantee 6
2.5 Why Capability Feedback? 6
2.6 Why Local Information? 7
2.7 How to Suppress Attack Traffic? 8
2.8 FlowPolice's Deployment 8
2.9 Is FlowPolice Scalable? 9

CHAPTER 3 SYSTEM DESIGN 11
3.1 System Overview 11
3.2 FlowPolice Packet Header 12
3.3 Secure the Request Channel 13
3.4 Unforgeable Capability 13
3.5 Flow Table 14
3.6 Rate Limiting Logic 16

CHAPTER 4 SECURITY ANALYSIS 24

CHAPTER 5 EVALUATION 27
5.1 Implementation 28
5.2 DDoS Attack Mitigation 30

CHAPTER 6 DISCUSSION 36

CHAPTER 7 RELATED WORK 37

CHAPTER 8 CONCLUSION 39

REFERENCES 40

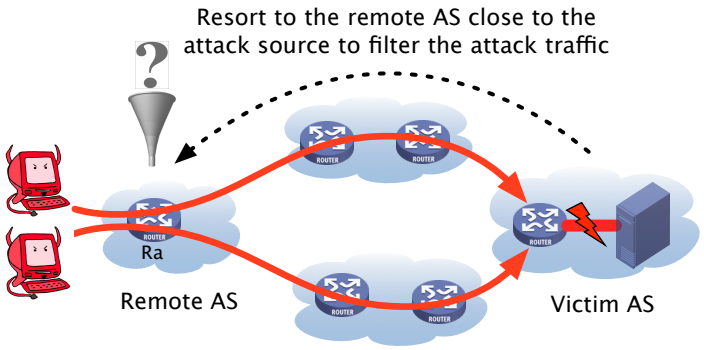
CHAPTER 1

INTRODUCTION

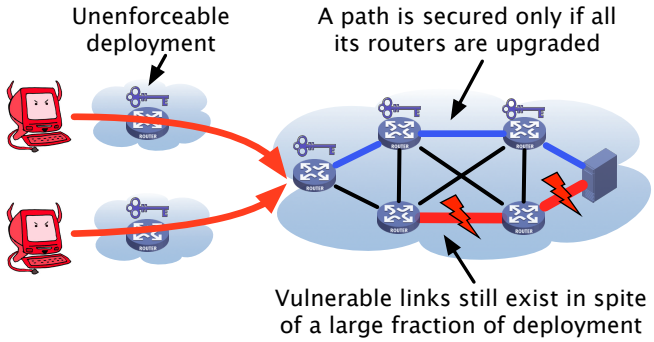
Distributed denial of service (DDoS) attacks are a serious threat to the Internet. In a flooding based DDoS attack, attackers paralyze Internet services by sending excessive packets to exhaust the servers' network bandwidth so as to stop legitimate users from accessing services. Over the past decades, the Internet has grown from a small research network to a piece of critical infrastructure with significant social, economic and political influence. Because of the newfound importance of the Internet, DDoS attacks can cause millions of dollars worth of financial losses. However, today's Internet is still fragile to DDoS attacks. According to reports from Arbor Networks [1] and Prolexic [2], DDoS attacks have grown in both size and frequency in 2014.

The meteoric rise in the significance of the Internet has brought forth a fundamental research challenge: How can we make the Internet more resilient to large scale DDoS attacks? Over a decade of research, there have been many proposals [3–19] to address this challenge. The previous approaches can be categorized into two major groups: network filtering based approaches (*e.g.,* [6, 10]) and capability based approaches (*e.g.,* [13, 14]). We discuss this distinction further in Chapter 7. Although these proposals are designed to let DDoS victims suppress attack traffic, unsatisfactory progress has been made on the real-world deployment due to prohibitive deployment hurdles [20, 21].

A significant deployment problem is that a deploying autonomous system (AS) yields no benefit until other AS also deploys the protocol. The inter-dependency among ASs creates a chicken-and-egg deployment problem. Specifically, most previous approaches require cooperation and coordination among ASs along the path to defend against attack. However, such large-scale cooperation is difficult to achieve in the Internet due to the large number of administrative domains. For instance, the filtering based approaches (*e.g.,* Pushback [6, 7], Passport [11]) rely on the ASs close to the attack source to block the attack traffic. However, these ASs may be beyond the control of the



(a) Filtering based approaches.



(b) Capability based approaches.

Figure 1.1: Deployment hurdles for filter-based and capability-based systems.

victim AS and may not even deploy the defense protocol (Figure 1.1(a)). Similarly, capability based schemes (*e.g.,* TVA [13], NetFence [14]) also require capability operation at remote ASs and need a large fraction of deployment to secure the entire AS (Figure 1.1(b)).

In this thesis, we propose a new DDoS defense mechanism, FlowPolice, to address the chicken-and-egg deployment problem. In particular, FlowPolice offers three desirable deployment features.

Local deployment: FlowPolice can immediately benefit the first deploying AS without any additional deployment at other ASs. The local deployment feature incentivizes deployment, even for the first AS.

Single deployment A FlowPolice deploying router can effectively suppress attack traffic even if congestion happens on a downstream link where the cryptographic protections of FlowPolice are not deployed. Therefore, if a link has a single FlowPolice router along each uplink path, that link is completely protected. The single deployment feature further reduces deployment requirements and maximizes deployment efficiency.

Minimal trust: The FlowPolice router needs to trust only values created by that router in making rate limiting decisions. Since FlowPolice places no trust in other routers or other ASs, FlowPolice is more robust than previous work with weaker trust models.

FlowPolice's novel design provides these three desirable deployment features while effectively defending DDoS attacks (Chapter 2). (i) In flooding based DDoS attacks, attackers maliciously inject excessive packets into the network to cause severe congestion. To suppress attack traffic, FlowPolice forces attackers to take *accountability* for the congestion by rate limiting. (ii) To learn users' congestion accountability, FlowPolice relies on unforgeable capability tags created by the deployed router itself. Therefore, FlowPolice can effectively stop attack traffic and ensure bandwidth share for legitimate users, while requiring no inter-router trust and deployment only at the victim AS.

The major contributions of this thesis are the design, implementation and evaluation of a new DDoS defense approach FlowPolice, which offers three desirable deployment features. We implement FlowPolice on Linux to demonstrate that FlowPolice's scalability. For instance, a single deployment on a commodity PC can effectively handle 100 million attack flows (Section 5.1). Furthermore, FlowPolice's packet processing overhead is small so that the deploying router introduces negligible networking overhead, Finally, we designed detailed packet-level simulation to demonstrate that FlowPolice can effectively mitigate large scale DDoS attacks.

CHAPTER 2

DESIGN RATIONALES

In this chapter, we describe the insights that lead to the design decisions of FlowPolice. We start with the problem space and design goals of FlowPolice.

2.1 Problem Space

Network flooding based DDoS attacks: FlowPolice focuses on mitigating flooding based DDoS attacks where attackers maliciously send excessive packets to exhaust the network bandwidth so as to stop legitimate users from getting adequate bandwidth. Resource exhaustion at other layers, such as the SYN flooding attack [22], are orthogonal to FlowPolice. Some application layer defense protocol can be incorporated with FlowPolice to provide even stronger defense (see more discussion in Chapter 6).

Adversary model: We consider strong adversaries that can compromise both end hosts and routers and adaptively adjust their attack strategies. For instance, they can launch on-off shrew attacks [23] to evade detection, collude to grant each other traffic capabilities, spoof addresses and can recruit large numbers of botnets to launch large scale DDoS attacks.

2.2 Design Goals

Deployable in the Internet architecture: The major design goal of FlowPolice is to be a defense system that is deployable in the current Internet architecture. To this end, we propose FlowPolice with three deployment features (*i.e.,* local deployment, single deployment and minimal trust) to minimize deployment hurdles.

Guaranteed fair bandwidth share: FlowPolice provably guarantees that legitimate users can get at least per-flow fair share for the bottleneck band-

width no matter what strategies attackers use. The flow means all aggregate traffic matching certain patterns. For instance, in the case where attackers try overflow the network resource of a public server, the worst case bandwidth share for legitimate users is determined by the ratio of the number of compromised hosts to the number of legitimate senders; *i.e.,* per-sender fairness is achievable. When some attackers do not strictly comply with the rate policing from FlowPolice, legitimate senders can get even more bandwidth share (see discussion in Chapter 5). Note that to ensure fairness, FlowPolice do not assume that IP address spoofing is eliminated. We will further elaborate the reason in Section 3.5.

Scalable and lightweight: FlowPolice is proposed to handle large scale DDoS attacks involving millions of attack flows, while using lightweight packet processing at relatively few nodes, resulting in negligible network overhead. When network operators are able to deploy FlowPolice at the upstream of all potential bottleneck links within their ASes, a single deployment in the AS is enough to secure the entire AS.

2.3 Enforce Congestion Accountability

Flooding based DDoS attacks represent an extreme form of network congestion where adversaries maliciously inject excessive packets into the network. A bottleneck router drops packets at random from all senders, regardless of whose traffic contributes more to the congestion. In other words, *congestion accountability* is not considered while dropping packets. Consequently, legitimate users are equally affected by congestion, even when such congestion is disproportionately caused by attackers. FlowPolice addresses this disproportionality by enforcing bandwidth limits on senders that consistently contribute to the congestion in the face of severe packet losses; *i.e.,* FlowPolice enforces congestion accountability in case of DDoS attacks.

Next we consider how to determine each participating flow's congestion accountability. Even in normal scenarios, congestion can also happen in the Internet. Legitimate senders (*e.g.,* TCP senders) adjust their rates based on the available bandwidth so that all participating flows are able to get their share. However, in DDoS attacks, the congestion is deliberately caused by attackers. They try to overflow the network and may not back off even

when they are aware that they are experiencing severe congestion. Therefore, one sender's congestion accountability should be determined by whether it adaptively changes its rate based on the current network condition and tries to relieve congestion when it happens.

FlowPolice uses the packet loss rate over an extended period of time (much larger than typical Internet RTTs) to quantify each sender's congestion accountability since senders that continue to transmit in spite of congestive losses are more accountable for the congestion. Note that FlowPolice does not police traffic based on a single or several packet losses to avoid false positives. FlowPolice enforces bandwidth limits on the senders whose packet loss rate exceeds a threshold, punishing senders that continuously contribute to congestion so as to throttle the unwanted traffic.

2.4 Fairness Guarantee

Another desirable feature for FlowPolice's policing policy is to ensure per-sender fair share at the bottleneck link. Specifically, FlowPolice keeps a *rate limiting window* for each flow in case of DDoS attacks so that no sender can send faster than its rate limiting window. Via an additive-increase/multiplicative-decrease (AIMD) update for the rate limiting windows, FlowPolice probably ensures that all senders' bandwidth share converges to fairness (see details in Section 3.6.4).

Next we discuss how FlowPolice learns each flow's loss rate so as to determine its congestion accountability.

2.5 Why Capability Feedback?

Inferring packet losses is the core problem in FlowPolice's design, since the further steps rely on correct packet loss information. However, in the Internet, it is challenging for an intermediate router to infer a flow's packet losses, since losses can happen downstream, and because return traffic will likely take a different path. Furthermore, one flow may cross ASs controlled by different network administrative units, limiting the usefulness of network management tools in loss-rate inference.

In FlowPolice, senders self-report their packet losses. As in re-feedback [24], in which each packet carries a congestion metric, FlowPolice adopts *capability feedback* to infer packet losses. Specifically, each FlowPolice router stamps a unique unforgeable capability tag on each traversing packet (Section 3.4). When the packet is received, the receiver returns the capability back to the sender (as in [13, 14, 25, 26]). Then the sender includes the capability tag in future packets to demonstrate to the FlowPolice router that the previous packet has been received. Otherwise, if the capability tag is not sent back to the FlowPolice router after a certain time interval (defined as detection period in Section 3.6.3), the router will consider that packet to have been lost. The router can use these reports to infer packet loss rates regardless of the number of downstream links the packet traverses.

2.6 Why Local Information?

FlowPolice infers packet losses only based on local information (capabilities created by the router that uses them). We believe that such a design removes FlowPolice's deployment hurdles while enhancing its robustness. Specifically, as the FlowPolice router infers packet loss using only capabilities created by that router, FlowPolice does not require coordination with other routers. Therefore, the first deployed ASs can yield immediate benefits, resolving the chicken-egg deployment problem. Furthermore, since FlowPolice can infer remote packet losses, a single deployed router can protect all its downstream links by suppressing attack traffic locally. Thus FlowPolice only requires a small deployment to protect a large fraction of the network.

Moreover, trusting only self-created tags improves system robustness. Specifically, FlowPolice capabilities are unforgeable since they are generated based on FlowPolice router's private key. Therefore, even though some routers or even ASes on the path are compromised, they cannot maliciously create valid capabilities to break the defense protocol.[1]

[1]The compromised routers may simply drop all packets. Such a security problem is known as the Internet hijack [27] or path validation, which is not the focus of this thesis. Several approaches have been proposed (such as [28–30]) to defend against such attacks.

2.7 How to Suppress Attack Traffic?

Congestion may happen at multiple links. However, it is difficult to determine these bottleneck links before attacks, and enforcing deployment at all these bottleneck links can be expensive. Therefore, FlowPolice provides a single deployment feature so that network operators do not have to know where the congestion happens to perform deployment and bottleneck routers do not even have to deploy FlowPolice.

To begin with, FlowPolice uses severe packet losses over an extended period of time to infer that some (remote) links are under DDoS attacks. Thus even though bottleneck links may be varying over time, as long as FlowPolice is placed at the upstream of them, FlowPolice is able to learn that these links are under attack. In case of DDoS attacks, FlowPolice polices the traffic as privileged packets and best-effort packets. Via the robust AIMD updating mechanism for each flow's rate limiting window, FlowPolice ensures that all flows can only send fair amounts of privileged packets so as not to flood the bottleneck link. Thus the bottleneck router can easily suppress the unwanted traffic by prioritizing privileged packets over best-effort packets. Note that FlowPolice places almost all the heavy lifting (*e.g.,* state maintenance, cryptography processing) at the upstream deploying router whereas the downstream bottleneck links only need to implement a simple prioritization mechanism, which can be relatively achieved, for instance, by QoS functionality. Furthermore, FlowPolice only polices the traffic that is accountable for the congestion so that the "unaccountable" traffic (*e.g.,* traffic not traversing the downstream bottleneck router or well behaved senders) will be not affected.

2.8 FlowPolice's Deployment

FlowPolice's single deployment feature allows network operators to secure their ASs with small amounts of deployment. One example of a small-scale deployment that can benefit a large number of potential victims is to deploy FlowPolice in a middle-service, which we call CloudPolice. A potential victim can obtain an IP address from CloudPolice. Then the victim serves DNS replies with the CloudPolice IP address so as to redirect all of the victim's

traffic to CloudPolice. CloudPolice itself would deploy within the Internet core, which increases its possibility to be located before any bottleneck link and reduce its probability of being overflowed.

When CloudPolice receives a FlowPolice packet, CloudPolice performs the flow policing functions of the FlowPolice router, then IP-in-IP encapsulates [31] the packet to the actual destination, and finally pulls the FlowPolice header to the outer IP header. For best effort traffic, CloudPolice simply IP-in-IP encapsulates the packets without further processing. As long as the bottleneck link gives sufficient priority to FlowPolice traffic, for example by allocating a percentage of the link to FlowPolice traffic, all traffic routed through CloudPolice will receive the benefits of FlowPolice. Depending on the victim's service provider's ingress filtering policies, the victim can either directly reply using the FlowPolice IP address, or can route its reply back through CloudPolice using IP-in-IP encapsulation.

2.9 Is FlowPolice Scalable?

When under DDoS attack, a FlowPolice router performs per-sender rate limiting to suppress attack traffic. Attackers may game the defense by exhausting the router table. The design of FlowPolice enables FlowPolice to scale up to deal with extremely large scale attacks. To begin with, attackers who recruit $\mathcal{N}_A$ compromised hosts can only consume up to $\mathcal{N}_A$ states; *i.e.,* attackers are not able to spoof IP addresses to fake excessive states in the router table. Furthermore, FlowPolice's single deployment feature gives network operators an opportunity to secure their ASs with a small number of deployed routers, given that the network operators know the in-AS topology and can perform in-network routing control via platforms such as RCP [32]. Ideally, a single deployed router can protect all the links within one AS. Therefore, network operators can scale up FlowPolice's performance, with reasonable cost, by either deploying FlowPolice on a high-end hardware router or implementing FlowPolice in a software router with large memories and high-speed CPUs. For instance, our implementation on a commodity PC (with 8GB memory and 3.4GHz CPUs) can effectively handle 100 million states (Section 5.1). It is clear that we can further increase FlowPolice's scalability with more advanced software router. Therefore, FlowPolice can effectively handle DDoS

attacks even involving millions of botnets.

Besides scalability, another good feature offered by the single deployment is that the packet processing delay is low since FlowPolice's cryptographic operations only need to be processed once. For instance, in our implementation (Section 5.1), per-packet processing overhead is $\sim 0.08 \mu s$, which transfers to around 150Gbps throughput for the 1500B packet size. Thus, deploying FlowPolice introduces negligible networking overhead.

CHAPTER 3

SYSTEM DESIGN

In this chapter, we describe the design of FlowPolice. We start with the system overview to construct a clear picture of FlowPolice's components. Then we detail the design of each component.

3.1 System Overview

The FlowPolice architecture is illustrated in Figure 3.1. Similar to the previous capability based approaches [13, 14], FlowPolice categorizes packets into three types: request packets, regular packets and legacy packets. Each sender needs to send a request packet to initiate a new *flow*. In this thesis, we use the term "flow" to indicate all traffic matching specific pattern (see details in Section 3.5). The regular packets carry the capabilities stamped by the FlowPolice router. The legacy packets are sent without carrying the FlowPolice packet header. Protecting legacy packets is an open problem as the original design of the Internet places too little emphasis on security [33]. FlowPolice only provide best-effort delivery for legacy packets.

The underlying packet processing logic of FlowPolice is as follows: when a new packet arrives, the FlowPolice router first determines its type. If it is a legacy packet, FlowPolice appends it to the legacy channel, which can only consume a small fraction of the link capacity (*e.g.,* less than 10%). If the packet is a request packet, FlowPolice add an initial capability in the packet header and forward the packet to the next hop. If the packet is a regular packet, the FlowPolice router checks the capability feedback carried by the packet and polices it based on its rate limiting decisions according to the *rate limiting logic* (Section 3.6). Meanwhile, the FlowPolice router updates information for the corresponding entry in the flow table (Section 3.6.1).

If the link directly connected to the FlowPolice router is not a bottleneck

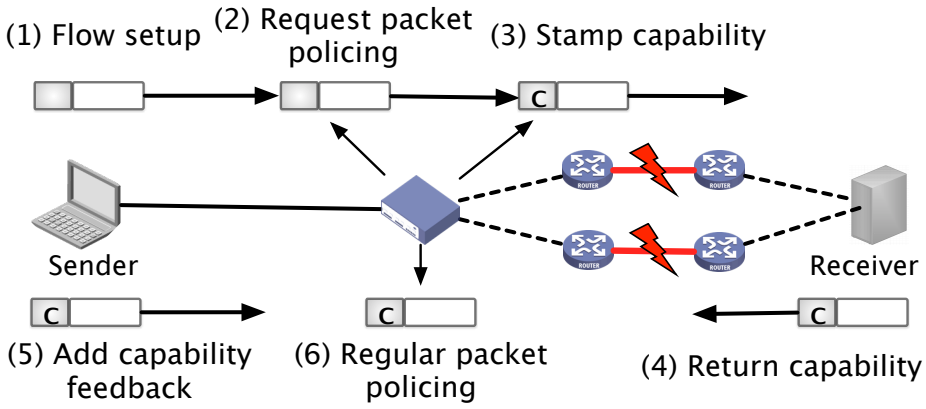


Figure 3.1: The architecture of FlowPolice. The deployed router enforces congestion accountability to suppress attack traffic. A single deployed router can protect all its downstream (potential) bottleneck links that implement a simple prioritization mechanism.

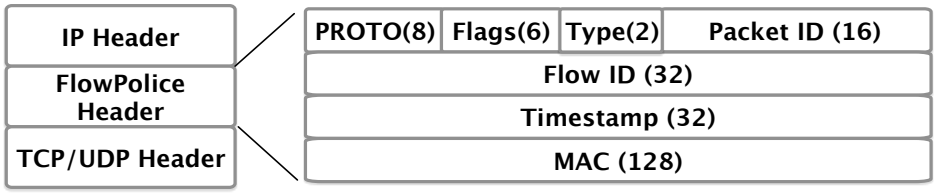


Figure 3.2: FlowPolice packet header format.

link, whereas congestion happens at some remote downstream links (as illustrated in Figure 3.1), then the bottleneck router can suppress the unwanted traffic by simply prioritizing the privileged packets (marked by the FlowPolice router) over the best effort packets (legacy packets and best-effort FlowPolice packets).

3.2 FlowPolice Packet Header

We place the FlowPolice header in the shim layer between the IP protocol and the transport layer protocol so that the applications need not be modified. The FlowPolice header is illustrated in Figure 3.2.

FlowPolice packets are assigned a specific IP protocol number. The FlowPolice header has a "next header" field (PROTO field in Figure 3.2) which is the real protocol number, as in IPv6 hop-by-hop and destination options headers. The Flags field is reserved. The Type field is used to decide the

packet type. 00 represents request packets (marked by the sender), 10 represents best-effort FlowPolice packets and 11 stands for privileged FlowPolice packets. The total size of the regular packet header is 28 bytes and the request packet needs to include a Portcullis [16] header (Section 3.3).

3.3 Secure the Request Channel

The request packets are vulnerable to DDoS attacks as well. Specifically, attackers can flood the request channel to stop legitimate users from getting valid capabilities. Such attack is known as the Denial of Capability (DoC) attack [16]. Two types of approaches have been proposed to address such an attack. One is implementing per-sender rate limiting [14] at the request channel so that each sender, including legitimate users, can finally get its share. The second approach is based on the proof-of-work schemas [16, 17]. For instance, the Portcullis [16] protocol requires each sender to solve a puzzle to send valid request packets. As adversaries have bounded computational power, they are not able to flood the request channel all the time.

FlowPolice adopts the Portcullis [16] protocol to secure the request channel for the sake of easy deployment. The per-send rate limiting approach relies on source spoof defense protocols such as Ingress Filter [34] and Passport [11]. However, the effectiveness of these protocols relies on universal deployment, which has not been implemented yet. Therefore, without ubiquitous deployment, Internet-wide source spoofing is still possible [20]. In contrast, Portcullis [16] is more deployment friendly. For instance, the puzzle can be obtained relatively easily via the exiting DNS services. Furthermore, the puzzle based solution can be integrated into the IPv6 deployment, as proposed in Mirage [21], which produces further deployment incentive.

3.4 Unforgeable Capability

The design of capability needs to meet the following requirements. (i) The capability cannot be forged by others. (ii) It can be easily generated so that the routers are not blocked while processing packets. (iii) The capability can be re-generated on the fly for the purpose of verification; thus, the router

does not need to store previously issued capability. The FlowPolice router issues its capability as follows:

$$\mathcal{C} = \mathcal{P}_{id} \mid\mid f \mid\mid ts \mid\mid MAC(\mathcal{P}_{id} \mid\mid f \mid\mid ts), \tag{3.1}$$

where $\mathcal{P}_{id}$ is packet ID, f is flow ID and ts is the timestamp. The meaning of $\mathcal{P}_{id}$ and ts will be described in the next section (Section 3.5). The Message Authentication Code (MAC) is computed using the router's private key. Since the capability incorporates a keyed MAC, no one besides the FlowPolice router can generate valid capabilities. Further, we use AES-128 based CBC-MAC for generating MAC due to its fast speed and availability at modern CPUs [35, 36]. Finally, the verification of capability is simply re-hashing the inputs carried in the packet header and checking whether the output is the same as the carried MAC.

3.5 Flow Table

In this section, we detail the flow table maintained by the FlowPolice router.

3.5.1 Flow entry creation

When the initial capability tagged in the request packet is fed back to the FlowPolice router, the router needs to determine whether to create a new flow entry in the flow table. If FlowPolice uses the traditional 5-tuple in the packet to identify a flow, the number of flow entries attackers can consume is determined by their computational power since we incorporate the Portcullis protocol [16] to secure the request channel. In other words, FlowPolice achieves per-computation fair share for the flow table among attackers and legitimate users.

In some cases, we can further reduce the number of states consumed by attackers in the flow table to the number of botnets they recruit. For instance, when a AS deploys FlowPolice to protect its public servers (*e.g.,* web servers), FlowPolice can keep per-sender state (*i.e.,* aggregating all traffic with same source IP address as one flow) for the traffic sending to its servers. This is because when attackers fake a source IP address not owned by their botnets

f	$\mathcal{T}_A$	$\mathcal{P}_{id}$	$\mathcal{R}_T$	$\mathcal{T}_C$	$\mathcal{S}_T$	$\mathcal{P}_R$	$\mathcal{W}_R$	$\mathcal{W}_V$	$\mathcal{L}_R$
64	32	16	1	1	32	32	32	128	64

Figure 3.3: Fields in each flow entry and the corresponding size (bits) for each field (total ∼50 bytes).

in a request packet, they can only receive the initial capability issued by the router and send it back to the FlowPolice router if the packet's destination is also compromised. Otherwise a legitimate receiver will return the capability to the spoofed IP address so that attackers cannot obtain the capability. Therefore, when the destination is known not be compromised,[1] the number of router states consumed by attackers is bounded by the number of botnets they have. Note that in the case where FlowPolice is deployed in middle-services, the service provider needs to be aware of the server addresses that they try to protect.

When source spoofing is eliminated (*e.g.,* Ingress Filter [34] has universal deployment), it is safe to keep-sender state for all traffic.

3.5.2 Flow entry

We define a flow as all the traffic matching certain patterns. (i) All packets whose capabilities have secure destination addresses and have the same source address are aggregated as one flow. (ii) All packets whose capabilities have insecure destination addresses and have the same destination address are aggregated as one flow. Each active flow is associated with a unique flow ID in the table. A simple way of assigning the flow ID is to use the source IP address as the flow ID if this flow's destinations are secure, and to use destination IP address as the flow ID if one flow's destination may be compromised. We assign 64 bits for the flow ID field.

The FlowPolice router maintains flow-level information in the flow table. As depicted in Figure 3.3, each flow entry is composed of 9 fields, but only 3 fields are updated relatively frequently based on packet arrival and the rest are updated once per detection period. The period $\mathcal{T}_A$ and packet ID $\mathcal{P}_{id}$ are used to create capabilities (Section 3.4). $\mathcal{R}_T$ is a one bit tag to determine

[1]In this case, if the public servers were compromised, attackers would not need to leverage flooding traffic to launch DDoS attacks any more. Thus we assume the servers are not compromised.

whether rate limiting should be applied for f. $\mathcal{T}_C$ indicates whether f is in its first period. $\mathcal{S}_T$ indicates the number of consecutive periods that f's loss rate is below the threshold. $\mathcal{P}_R$ indicates the number of received packets from f. $\mathcal{W}_R$ decides the maximum number of privileged packets allowed for f. The verification window $\mathcal{W}_V$ is used to infer packet losses. $\mathcal{L}_R$ is the packet loss rate for f. In Section 3.6, we will describe how to populate the flow table and articulate how to use these flow-level information to make rate limiting decisions.

3.6 Rate Limiting Logic

FlowPolice makes rate limiting decisions based on inferred packet losses. In normal scenarios (without DDoS attacks), FlowPolice does not enforce rate limiting so that each flow can get its share of the bandwidth. However, when FlowPolice infers severe packet losses, it will execute per-flow rate limiting to suppress attack traffic.

3.6.1 Populating the flow table

Before diving into the detail of the rate limiting algorithm, we first describe how to populate the flow table. Assume at time ts, a new flow f is established (puzzle solution is verified). All fields of the newly created flow entry are initialized to be zero. The router first updates $\mathcal{T}_A$ as ts. Then it increases both $\mathcal{P}_R$ and $\mathcal{P}_{id}$ by one, creates a new capability specific to the new $\mathcal{P}_{id}$, adds the capability to the packet header and forwards the packet to next hop.

When the router receives a regular packet from f, it first increases $\mathcal{P}_R$ by one. Then it performs hash verification to determine whether the carried capability is valid. Unverified packets are dropped. Otherwise, based on the verified capability, the router updates the verification window $\mathcal{W}_V$ to keep track of the received feedbacks. Further, the router may also create a new capability to replace the old one. We defer the details for updating $\mathcal{W}_V$ and generating capabilities in Section 3.6.2.

When kth detection period ends and k+1th period starts, FlowPolice needs to perform flow analysis (*e.g.,* learn loss rates) and updates the flow table as

well. The router detects a new period for f when it receives the first packet from f in k+1th period. Specifically, when a packet arrives, if $\mathcal{T}_A$ in its flow entry is more than one period older than the packet arrival time ts (*i.e.,* $\mathcal{T}_A + \mathcal{D}_p < ts$, where $\mathcal{D}_p$ is the length of detection period), the router realizes that the current packet is the first one received in k+1th period. Then the router updates $\mathcal{T}_A$ as ts, and performs the following updates in order.

Period counter: If $\mathcal{T}_C > 0$, do not update it. Otherwise, increase $\mathcal{T}_C$ by one.

Rate limiting: Update $\mathcal{W}_R$, $\mathcal{R}_T$, $\mathcal{S}_T$ and $\mathcal{L}_R$ according to the Algorithm 1 (§3.6.3).

Reset: Set $\mathcal{W}_V$, $\mathcal{P}_R$ and $\mathcal{P}_{id}$ to zero.

3.6.2 Inferring packet losses

The FlowPolice router uses capability feedback loop to infer packet losses, especially the losses caused by the remote bottleneck links. Note that the receiver needs to return the capabilities back to the sender so that the sender can present these capabilities to the router to prove the delivery of packets. Thus if the detection period length is at least two times the typical Internet RTTs, the capabilities stamped in the first half of each period should be received by the router by the end of the period. In contrast, if some packets carrying capabilities are lost somewhere on their way to the destination, the receiver will not receive these packets so it cannot return the capabilities back to the sender. As a result, the router cannot receive those lost capability feedbacks as well. Therefore, by monitoring the capability feedback loop, the FlowPolice router can infer remote packet losses.

Next we describe how to create new capabilities. For each of the first K_{th} packets that arrive at during the first half of each period (*i.e.,* the arrive time $ts < \mathcal{T}_A + \mathcal{D}_p/2$), the router increases P_{id} in the flow entry by one, creates a new capability specific to the new $\mathcal{P}_{id}$ via Equation (3.1), stamps the new capability in the header to replace the old one and forwards the packet. Then the router waits for these K_{th} unique capabilities till the end of the period. If a flow only sends K_0 packets and $K_0 < K_{th}$, the router will just create K_0 capabilities for the flow.

We add two constraints for generating new capabilities: only creating new

capabilities during the first half of the period and generating at most K_{th} capabilities in each period. The first constraint is because we need to allow sufficient time (at least half of $\mathcal{D}_p$) for each capability to return back to the router so that the unreceived capabilities are lost rather than delayed. The second constraint is added to save memory usage for the capability feedback verification, as described below.

Assume the router generates K_{th} distinguished capabilities with packet ID ranging from $[1, K_{th}]$ during the first half of the kth period. Each time a new packet carrying valid capability arrives during this period, the router checks the packet ID in its capability to determine which previously stamped packet has been received by the receiver. We implement $\mathcal{W}_V$ as a window with K_{th} bits. Initially all the bits are set as zero. If a packet carrying a capability with packet ID i is received, the router sets the ith bit in $\mathcal{W}_V$ as one. Therefore, if by the end of the period, all the K_{th} bits in $\mathcal{W}_V$ are one, the router can verify that all the stamped packets during the first half of the period have been received by the receiver. Otherwise, the packets indicated by the zero bits in $\mathcal{W}_V$ are lost. To avoid capability reuse attack, the router only accepts the capability feedbacks issued within the current period (see more details in Chapter 4).

Note that each issued capability consumes one bit in $\mathcal{W}_V$. To save memory usage and meanwhile be able to learn the statistical packet loss rate, we set the threshold $K_{th} = 128$ so that $\mathcal{W}_R$ consumes only 16 bytes space, whereas it is large enough to avoid statistical bias. For the rest of the packets received during the first half of the period (besides the first K_{th} ones) and the packets received during the second half, the router stops issuing new capabilities and simply performs capability verification to update the $\mathcal{W}_V$ correspondingly.

The statistical packet loss rate of flow f during the kth period can be obtained via $\mathcal{P}_{id}$ and $\mathcal{W}_V$. Note the number of capabilities generated for f is recorded in $\mathcal{P}_{id}$ of the flow entry and $\mathcal{P}_{id} <= K_{th}$. Thus the inferred packet loss rate is $\frac{V_0}{\mathcal{P}_{id}}$, where V_0 is the number of unverified bits (zero bits) in $\mathcal{W}_V$.

3.6.3 Rate limiting algorithm

FlowPolice makes the rate limiting decision for f based its packet loss rate $\mathcal{L}_R$. One design detail is that during the first detection period, the router

cannot make a rate limiting decision for f since it has not yet learned its packet loss rate. However, if FlowPolice does not place rate limiting for all flows during the first period, attackers can compromise the defense by continuing to generate new flows so that these new flows are within the initial period and can get a free pass.

To address such attacks, FlowPolice sets the packets from flows that are in their first period ($\mathcal{T}_C$ is zero) as the best-effort FlowPolice packets by properly marking the FlowPolice packet header. Note that the bottleneck router (either the FlowPolice router or the remote one) prioritizes the privilege packets over the best effort packets. Therefore, such a design offers two good features. First, in normal scenarios without DDoS attacks, the arrived packets, including the best-effort FlowPolice packets, can be processed timely since the network is often over-provisioned [37] so that the router has enough capability to deal will all arrived packets. Therefore, FlowPolice does not have negative effects in a normal situation. Second, in case of DDoS attacks, attackers cannot exhaust the bandwidth via excessive new flows since their packets from the fresh flows are processed with low priority. The only effect of continuing to generate new attack flows is that legitimated flows that are in their initial period may experience large queuing delay. However, the problem has been downgraded from denial of service to low quality of service.

Starting from the second detection period, the router uses the statistical loss rate to make rate limiting decisions. The rate limiting algorithm is in Algorithm 1. In particular, FlowPolice adopts the metric *packetLoss* to determine whether f's rate should be limited. When obtaining *packetLoss* for f, we consider its previous packet losses as well as the recently learned loss rate. Such a design is used to defend against the on-off shrew attack [23] (see detailed analysis in §4). If *packetLoss* is larger than the pre-defined thresholds $L^{\downarrow}_{Th}$ (§3.6.5), FlowPolice classifies f as a maliciously behaved flow. Therefore, FlowPolice enables its rate limiting and multiplicatively reduces its $\mathcal{W}_R$. Otherwise, FlowPolice believes that f is well-behaved and turns off its rate limiting correspondingly.

Based on the rate limiting decisions, executing the rate limiting is simply marking all the newly arrived packets as best-effort FlowPolice packets when $\mathcal{P}_R > \mathcal{W}_R$ and $\mathcal{R}_T = 1$. FlowPolice turns off the rate limiting after f's loss rate has been below the threshold for $\mathcal{S}_{Th}$ consecutive periods, indicating that f is not experiencing severe congestion so that FlowPolice does not need to

police traffic for f.

Algorithm 1: Rate limiting algorithm.

```
1  begin
2      for each arrived packet P of flow f do
           /* first detection period                                  */
3          if T_C < 1 then
4              Mark P as a best-effort FlowPolice packet;
5          else
6              if P is the 1st packet of f in kth period then
7                  RateLimit();
8  RateLimit() begin
       /* recently learnt packet losses in kth period               */
9      recentLoss ← V_0 / P_id;
10     if recentLoss < L_Th^↑ then
11         S_T ← S_T + 1;
12     else
13         S_T ← 0;
       /* stop policing after S_Th consecutive low loss periods     */
14     if S_T > S_Th then
15         R_T ← 0;    W_R ← R_W;
16     else
17         R_T ← 1;
       /* policing policy                                           */
18     if R_T then
           /* consider history packet losses                        */
19         packetLoss ← (1 − λ) ∗ recentLoss + λ ∗ L_R;
           /* severe packet losses                                  */
20         if packetLoss > L_Th^↓ then
               /* Half the rate limiting window                     */
21             W_R ← (1 − β) · W_R;
22         else if packetLoss < L_Th^↑ then
               /* set W_R as the received packet counter            */
23             W_R ← W_R + △;
           /* update L_R                                            */
24         L_R ← packetLoss;
```

To sum up, FlowPolice's rate limiting algorithm ensures that in normal scenarios, all flows can effectively utilize the bandwidth via privileged packets by turning off their rate limiting tags. In case of DDoS attacks, FlowPolice polices the traffic via the AIMD updates for each flow's rate limiting window to resolve the congestion. Each participating flow has to comply with the rate limiting and adjust its rate accordingly so as to obtain its fair share (detailed in §3.6.4). Otherwise, the rate limiting window of misbehaved flows (*e.g.,* keep sending large volume of traffic in spite of severe losses) will keep reducing and most of their traffic is marked as best-effort. Therefore, the bottleneck router (either the FlowPolice router or a remote legacy router) can easily thwart the unwanted traffic by implementing a simple prioritization mechanism.

3.6.4 Fair bandwidth share guarantee

In this section, we prove that the rate limiting algorithm ensures per-flow fairness. We start with the following lemma stating that $\mathcal{R}_W$ for each flow converges to the fair share of the bottleneck link.

Lemma 1. *Given that $\mathcal{N}_L$ legitimate flows and $\mathcal{N}_A$ attack flows share the bottleneck link with capacity C, the rate limiting window $\mathcal{R}_W$ for each flow converges to $O(\frac{C}{\mathcal{N}_L+\mathcal{N}_A})$ after enough rounds of AIMD updates.*

Proof. This lemma is borrowed from the theoretical analysis in [38]. Specifically, when N users share a link with bounded capacity and each user adjusts its rate based on link feedback via the AIMD mechanism (*i.e.,* negative feedback causes multiplicative decrease to the rate and positive feedback causes additive increase to the rate), all users' rates will finally converge to fairness. In the rate limiting algorithm, FlowPolice updates each flow's $\mathcal{R}_W$ based on packet losses via the AIMD mechanism. Therefore, each flow's $\mathcal{R}_W$ will eventually converge to fairness. □

Given Lemma 1, we obtain the following lemma.

Lemma 2. *In the steady state, each attack flow can obtain at most $\frac{(1+L_{Th})C}{\mathcal{N}_L+\mathcal{N}_A}$ share at the bottleneck link.*

Proof. Note that FlowPolice allows a maximum $L_{Th}^{\downarrow}$ loss rate before further cutting the rate limiting window. Thus the optimal strategy for an attacker flow is to send no more than $1+L_{Th}^{\downarrow}$ times its rate limiting window, *i.e.,* to strictly comply with FlowPolice's rate limiting. Otherwise, its share will be further reduced. □

Combing the above two lemmas, we obtain the following theorem.

Theorem 1. *In the steady state, each legitimate flow can obtain at least* $\frac{(1+L_{Th}^{\downarrow})C}{\mathcal{N}_L+\mathcal{N}_A}$ *share at the bottleneck link, given that its transport protocol can fully utilize the allowed bandwidth.*

Proof. Given the fairness of rate limiting windows among all users, the fair bandwidth share for a legitimate flow is (*i.e.,* $\frac{(1+L_{Th}^{\downarrow})C}{\mathcal{N}_L+\mathcal{N}_A}$) if its transport protocol can fully utilize the allowed bandwidth. Note that the per-flow bandwidth share is lower-bound for legitimate flows. In the case where attackers do not use their optimal strategy (*e.g.,* sending flat rates), legitimate flows can get more bandwidth share than the per-flow fair share (see evaluation results in Chapter 5). □

3.6.5 Parameter discussions

FlowPolice's design has several important parameters: the length of detection period $\mathcal{D}_p$, the rate limiting factor β, the additive value $\triangle$, the two packet loss rate thresholds $L_{Th}^{\uparrow}$ and $L_{Th}^{\downarrow}$, the weight for historical packet loss λ and the S_{Th} for stopping traffic policing. We now discuss the reasoning for the parameter choices.

$\mathcal{D}_p$: The length of detection period should be at least two times the typical RTTs of the Internet so that the capabilities can have sufficient time to return back to the router. We tested the RTTs from our lab for top visited websites [39] located on different continents. Most of the RTTs are within 50∼200ms and the largest one is less than 400ms. Further, relatively longer detection period helps reduce statistical bias. However, $\mathcal{D}_p$ cannot be too long so that FlowPolice cannot react to attacks agilely. To balance these factors, 2∼4s is a reasonable choice for the period length.

β and $\triangle$: Note that the packet loss rate is a statistical result obtained over an extended period of time (much longer than the RTTs). There are two

cases in which one flow has high loss rate. The first is when it is experiencing a severe congestion. The second is when the flow does not comply with FlowPolice's rate limiting and sends more packets than its $\mathcal{R}_W$. Under both cases, FlowPolice needs to aggressively limit its traffic so that we set $\beta = 0.5$ in our algorithm. $\triangle$ is the additive value for updating the window. As $\mathcal{D}_p$ is several times larger than the typical RTTs, we set $\triangle = 3$.

$L^{\downarrow}_{Th}$ and $L^{\uparrow}_{Th}$: The choice of $L^{\downarrow}_{Th}$ should be larger than the normal packet loss rates. According to the previous measurements [40, 41], we set $L^{\downarrow}_{Th} = 5\%$. $L^{\uparrow}_{Th}$ should be small enough to indicate the current congestion is not that severe so that users can slightly increase their rate. We set $L^{\uparrow}_{Th} = 1\%$. In fact, based on their own network designs (*e.g.,* the over-provisioning ratio and the extent of traffic burstiness), network operators can have customized settings on $L^{\uparrow}_{Th}$ and $L^{\downarrow}_{Th}$.

λ: The value of λ represents how much credit FlowPolice will give on the previous packet losses. To defend against the on-off shrew attacks (Chapter 4), FlowPolice gives high weight to the packet loss history by setting $\lambda = 0.5$. Therefore, once a flow misbehaves, it will have a bad reputation for a while.

S_{Th}: If f's loss rate is below $L^{\uparrow}_{Th}$ for $\mathcal{S}_{th}$ consecutive periods, FlowPolice stops policing its traffic. Therefore, S_{th} should be long enough (*e.g.,* $\sim$10$-$100) to indicate that the congestion has been resolved. In other words, FlowPolice will stay in active mode until attackers stop launching DDoS attacks.

CHAPTER 4

SECURITY ANALYSIS

In this chapter, we perform security analysis to demonstrate FlowPolice's robustness. Specifically, we show how smart attackers may adjust their strategies to break the defense and demonstrate that FlowPolice is resilient to their attacks. FlowPolice uses the capability feedback to suppress illegal traffic. Therefore, attackers who want to launch DDoS attacks have to break the capability feedback loop to convince FlowPolice that they are sending legal traffic. We enumerate the strategies that attackers may adopt to break the defense.

Capability forgery: The straightforward strategy for attackers is to forge capabilities to falsely tell the router that the lost packets have been received. The design of the capability (Section 3.4) ensures that no one besides the FlowPolice router can create valid capabilities since they are incorporated with a keyed MAC. As long as the private key is secure, no capabilities can be forged.

Capability misuse: Attackers may try to misuse capabilities to tell the router that the lost packets were received. They can misuse capabilities in the following two ways. First, they try to use the capability to verify the delivery of a packet with a different packet ID. Second, they try to use the capability with the same packet ID but issued in previous periods to verify the delivery of a packet sent in the current period. Specifically, in kth period, we denote the capability issued with packet ID i by $\mathcal{C}(i,k)$, which is carried by the packet $\mathcal{P}(i,k)$. If $\mathcal{C}(i,k)$ is fed back to the router by the end of kth period, FlowPolice confirms that $\mathcal{P}(i,k)$ has been successfully received by the receiver. In the first misuse case, attackers use $\mathcal{C}(i,k)$ to verify the delivery of $\mathcal{P}(j,k)$. Clearly, this is infeasible as $\mathcal{C}(i,k)$ can only be used to verify the delivery of $\mathcal{P}(i,k)$. In the second case, attackers use $\mathcal{C}(i,k)$ (or even the capabilities issued earlier than the kth period) to verify the delivery of $\mathcal{P}(i,k{+}1)$. Again, FlowPolice will discard the capability because the router

only accepts capabilities issued within the current period. To sum up, the uniqueness of the combination of the detection period and packet ID ensures that no capabilities can be reused. If the some packets are lost, there is no way to deceive the router that they were not.

On-off shrew attack: The on-off shrew attack is also known as the low-rate TCP attack [23] in which attackers set up periodic on-off "square-wave" flows to exhausts the network resources. The attack exploits the TCP retransmission timeout mechanism and periodically causes packet losses for legitimate TCP flows. Attackers properly tune the attack period so that each time legitimate TCP flows recover from the timeouts, they will face another attack peak, which forces them to enter even longer timeouts. Since FlowPolice uses the packet losses inferred in the previous period to determine the rate limiting for the current period, adversaries have the incentive to launch on-off attack to evade the rate limiting. Specifically, in kth period, the attack flows send excessive traffic to overload the links. Then in k+1th period, FlowPolice will suppress these flows due to the severe packet losses. However, smart attackers can simply stop sending in period k+1. As a result, the router will detect no packet losses for them in the k+1th period so that it will turn off their rate limiting tags for the k+2th period. Consequently, attackers can re-launch attacks in period k+2. Such an on-off cycle repeats, and legitimate flows are throttled to small throughput. The design of FlowPolice has considered such a smart strategy. Note that when the router makes rate limiting decisions in Algorithm 1, it uses the metric $packetLoss$ which incorporates the previous packet losses. And FlowPolice gives high credit to the history via proper parameter choice. As a result, as long as their flows have severe packet losses in period k, attackers cannot "clear" the history even if they stop sending in the k+1th period. As a result, attackers cannot re-launch an attack in period k+1.

Flow regeneration attack: Attackers may regenerate new flows once their previous attack flows are detected; *i.e.,* after several detection periods, their rate limiting windows are small. FlowPolice is resilient to such attacks as well. On one hand, each time attackers initiate a new flow, they have to solve a puzzle (Section 3.3). Therefore, the number of flows attackers can initiate is bounded. Furthermore, as each new generated flow has to go through rate limiting as well (Section 3.6.3), creating new flows actually makes no difference.

Other strategies: Other strategies (such as collusion between senders and receivers) cannot break the defense as well since the design of FlowPolice makes no further assumptions on attackers.

CHAPTER 5

EVALUATION

In this chapter, we describe the evaluation of FlowPolice. We implement FlowPolice on a software router to demonstrate that FlowPolice introduces small packet processing overhead and can scale up to deal with very large scale DDoS attacks involving millions of attack flows. We further evaluate FlowPolice's packet forwarding performance via Click [42] implementation on our testbed. Finally, we carefully design detailed packet-level simulation on ns-3 [43] to prove that FlowPolice is effective to mitigate large scale DDoS attacks.

To evaluate FlowPolice, we also compare FlowPolice with previous proposed approaches. Arguably, all these approaches are effective under certain assumptions and deployment requirements. Without loss of generality, we choose one of the previous proposed approaches, NetFence [14], as our benchmark. FlowPolice's capability feedback design is inspired by NetFence's congestion policing feedback, whereas NetFence has different ways of inferring congestion. Specifically, FlowPolice relies on the self-created capabilities but NetFence relies on the congestion markings by the bottleneck links. Another difference is that FlowPolice polices bandwidth share based on congestion accountability whereas NetFence ensures per-sender fair for the bottleneck link. In our evaluation, we demonstrate that FlowPolice improves NetFence in the following two perspectives. (i) FlowPolice introduces smaller per-packet processing overhead. (ii) FlowPolice allows legitimate senders to potentially achieve (almost) their desired throughput, rather than just per-sender fair share when attackers do not strictly comply with the rate limiting enforced by FlowPolice.

5.1 Implementation

To evaluate FlowPolice's packet processing overhead, we implement FlowPolice on a software router equipped with 3.4GHz Intel i7 processors and 8GB DDR3 RAM. To evaluate FlowPolice's packet forwarding performance, we set up a two-system testbed. The first system `A` sends traffic to the second system `B` which serves as both the software router and the receiver. Both systems are running Ubuntu kernel version 3.2.0-64. We add FlowPolice's packet processing logic into `B`'s Click implementation, whereas `A` runs the original Click software. `A` and `B` are connected via an Ethernet switch with 1Gbps ports. Click is running at the kernel mode.

In our implementation, we use the CBC-AES (with a 128 bits key) based on the Intel AES-NI library to compute the MACs in the capabilities due to its fast speed and available support on our software router [35]. For instance, our implementation on a single i7 core can support AES encryption at the rate of $\sim$4 cycles per byte and decryption at rate of $\sim$0.7 cycle per byte. Further optimization is possible [44].

5.1.1 Scalability

As discussed in Section 3.5, the size for one flow entry is $\sim$50 bytes. Thus, even though the FlowPolice router maintains a flow table for 100 million flows, the memory size is only $\sim$5GB, which can be effectively managed by our software router with 8G memory. We show the per-packet processing overheads for three table sizes (1, 10 and 100 million entries) in Table 5.1.

For a flow table with 1 million entries, the overall per-packet processing overheads introduced by FlowPolice for request packets and regular packets are $\sim 0.2\mu s$ and $\sim 0.075\mu s$, respectively.[1] The request packet has larger overhead since the software router needs to assign memory for the new flow entry, which is time-consuming. As request packets are sent to initiate new flows and make up only a small fraction of total packets, they cannot become the bottleneck. Note that the overall per-packet processing overhead is composed of two major parts: flow table update time and AES operation overhead. In Table 5.1, we also explicitly list the flow table update overhead,

[1] In Table 5.1, we do not show the overhead for puzzle verification in request packets' processing overhead. Based on [16], such overhead is less than $1\mu s$.

Table 5.1: FlowPolice's per-packet processing overhead (μs) on our testbed.

Flow table size	Packet type	Table update overhead	Overall overhead
1 million	Request	~0.18	~0.20
	Regular	~0.047	~0.075
10 million	Request	~0.18	~0.20
	Regular	~0.050	~0.078
100 million	Request	~0.18	~0.20
	Regular	~0.052	~0.080

which makes up over 60% of the overall overhead.

From the results, it is clear that the per-packet processing overheads for 1 and 100 million flow table size are almost the same. Thus FlowPolice can effectively scale up to deal with DDoS attacks with millions of attack flows. Furthermore, the $\sim 0.08\mu s$ per-packet processing overhead will transfer to ~150Gbps throughput for 1500B packets. Thus the deployed router can still support high speed Ethernet.

As our benchmark, we compare FlowPolice's packet processing overhead with NetFence [14]. Based on its implementation, NetFence's per-packet processing overhead is around $1.3\mu s$. This is because their software router cannot support AES operation at line speed. However, even assuming that NetFence is deployed on routers with AES operation supported at line speed, it may still introduce larger overhead that FlowPolice. To begin with, NetFence requires cryptography processing at both the access router and the bottleneck router (may be more than one). However, FlowPolice only introduces one-time overhead at the deploying router. Furthermore, NetFence needs around 2GB memory at the access router to keep 1 million rate limiters, whereas FlowPolice only requires ~50MB. As memory accessing is more time-consuming, NetFence may have larger packet processing overhead.

5.1.2 Packet forwarding performance

In this section, we evaluate FlowPolice's packet forwarding performance. In particular, we examine the throughput (the bandwidth utilized by whole

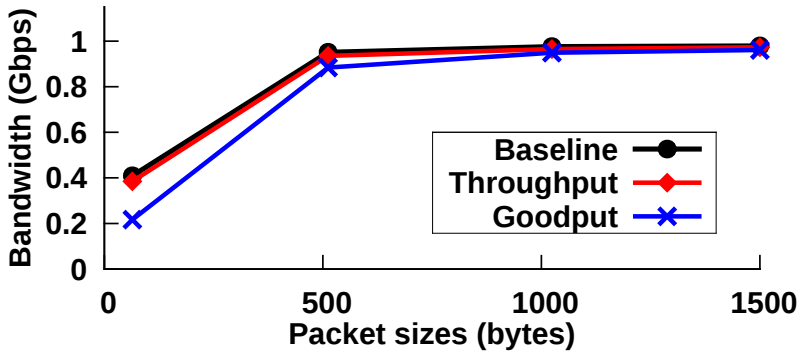


Figure 5.1: Packet forwarding performance.

packets) and goodput (the bandwidth used to deliver the payload of the packets, excluding the FlowPolice header). System `A` sends traffic at its full speed (up to 1Gbps) with different packet sizes to system `B`. We perform measurements at `B` to learn the throughput and goodput.

The results are illustrated in Figure 5.1. The baseline experiments are performed without enabling FlowPolice. We consider 4 packet sizes (64B, 512B, 1024B and 1500B, including the FlowPolice header) in the experiments. Note that for the smallest packet size (*i.e.,* 64B), the baseline throughput is much less than the available link capacity (1Gbps). The decline is mainly caused by the limitation of the Click router's throughput; *i.e.,* although the network links still have extra bandwidth, the Click router cannot process packets faster. It is clear that FlowPolice's throughput is close to the baseline due to its small packet processing overhead. As FlowPolice's header size is fixed (28B), the goodput is close to the throughput for larger packet sizes.

5.2 DDoS Attack Mitigation

In this section, we evaluate the effectiveness of FlowPolice for mitigating DDoS attacks via detailed packet-level simulations on ns-3 [43].

5.2.1 Methodology

We consider three representative DDoS attack scenarios in our evaluation to thoroughly investigate FlowPolice's performance (illustrated in Figure 5.2). In attack case `(a)`, the bottleneck link directly connects to the FlowPolice

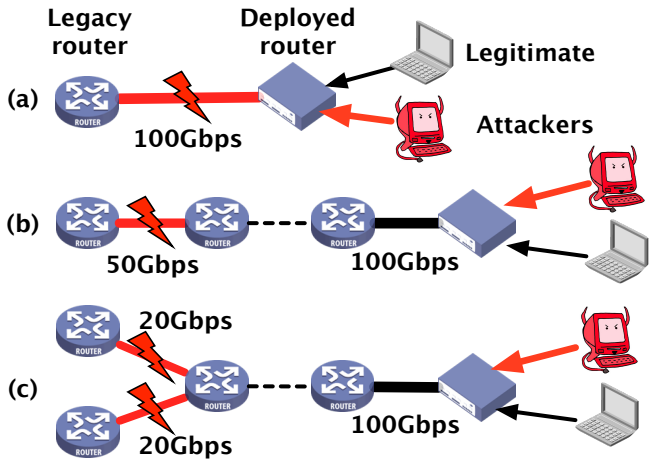


Figure 5.2: Attack cases in our evaluation.

router. In scenario (b), the bottleneck link is located at one of the FlowPolice router's downstream links. The third scenario (c) focuses on the case where multiple remote bottleneck links exist. The second and third settings are designed to demonstrate the effectiveness of FlowPolice's single deployment.

We desire to emulate the real-world DDoS attacks in which up to millions of attack flows try to flood a link whose capacity is on the order of gigabytes per second. However, the existing packet-level simulators or emulators, such as ns-2 [45], ns-3 and Mininet [46], will take prohibitively long to emulate extremely large scale DDoS attacks. Assume that the simulator can process one million packets per second and that one million attack flows, each sending at 5Mbps, try to flood a 10Gbps link. Even if we set the packet size as the maximum allowed size 1500B, it will take the simulator around 140 hours to simulate just one second of the attack.

To resolve the problem, we adopt the similar approach used in NetFence [14]. Specifically, we fix the number of nodes (both attackers and legitimate users) and scale down the link capacity to simulate the large scale attacks. For instance, by varying the link capacity from 50Mbps to 500Mbps, we are able to simulate the attack scenarios where $100K$ to 1 million attackers try to flood a 100Gbps link.

We evaluate FlowPolice using two performance metrics: flow completion time (FCT) and average throughput. To evaluate the first metric, each legitimate sender transmits a 2MB file to its destination and we use the file transferring time as the FCT. Such traffic represents the common web browsing

traffic where a typical web page size is around 2MB [47]. We put emphasis on FCT for this experiment setting since latency is the more important performance metric for short flows [48]. To learn the average throughput, legitimate users are configured to set up long-lived TCP flows. Such case corresponds to the Internet background traffic such as the WAN workloads [37, 49]. Under such scenarios, throughput is the right metric as these flows carry huge volumes of traffic and typically are latency insensitive.

5.2.2 Throughput evaluation

As proved in Section 3.6.4, FlowPolice ensures that attackers cannot gain more than their fair share at the bottleneck even with their optimal strategies. In reality, it may be difficult for attackers to learn the exact maximum allowed rates since they are unaware of all these rate-limiting parameters and how much traffic the bottleneck link can handle. Therefore, attackers need to probe the available network resources and adjust their rates accordingly. In our evaluation, we consider three representative strategies for attackers. The first strategy is a hypothetical in which that attackers can know their exact allowed rates. The major purpose of considering this strategy is to align our evaluation with our theoretical analysis. In the second strategy, attackers probe the available bandwidth so as to comply with their rate limit. This strategy represents the real-world optimal strategy for attackers. In the third strategy, attackers do not comply with the rate limiting enforced by FlowPolice (*e.g.,* sending flat rate in spite of packet losses). Such a strategy is effective and common in practice when defense protocols do not incorporate congestion accountability in the traffic policing.

As wide area networks often have $2-3\times$ bandwidth over-provisioning to tolerate traffic burstiness [37], we set the average traffic volume of legitimate users as half of the link capacity in our evaluation; *i.e.,* the desired throughput for all legitimate senders is half of the capacity. However, the attack traffic volume can be as much as $2\times$ the link capacity. The number of legitimate senders is $\sim$10% of the attacker number as attackers may recruit large numbers of botnets. Figure 5.4 shows the evaluation results. For simplicity, we use the TCP protocol for legitimate users' transport protocol, whereas attackers can have their self-defined transport protocol. To perform

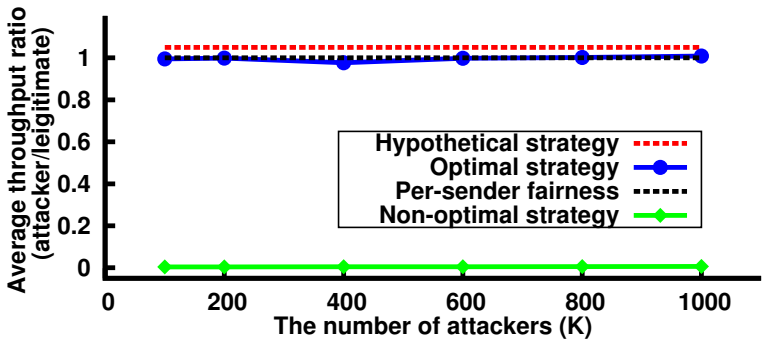


Figure 5.3: Throughput ratio for attackers. When attackers adopt their optimal strategy, their throughput share converges to fairness. However, if attackers do not comply with FlowPolice's rate limiting, their throughput is throttled to almost zero. The hypothetical margin is determined by $L_{Th}^{\downarrow}$.

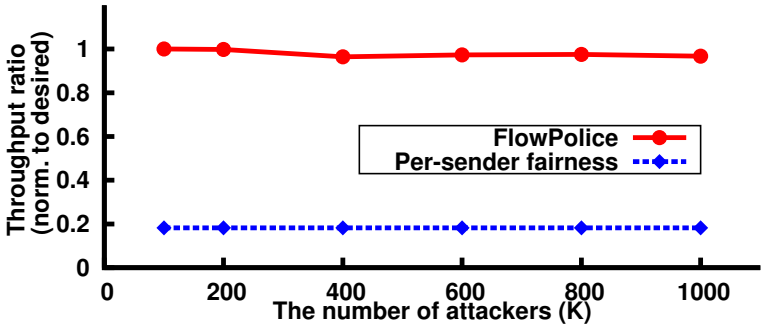


Figure 5.4: Throughput ratio for legitimate users when attackers do not comply with FlowPolice's rate limiting.

fair comparison with NetFence, we make the same assumption as NetFence in the evaluation: source spoof is eliminated. However, the design of FlowPolice does not assume that source spoofing is eliminated.

The results are illustrated in Figure 5.3. It is clear that the optimal throughput for attackers is the per-sender fair share. If attackers do not strictly comply with the rate limiting enforced by FlowPolice, their throughput share is reduced to almost zero. In such a case, legitimate senders can achieve almost their desired throughput as illustrated in Figure 5.4. Note that we evaluate all legitimate senders together rather than explicitly evaluate each sender's throughput. This is because FlowPolice is responsible for suppressing attack traffic so that legitimate senders are not affected by the flooding attack. However, FlowPolice does not explicitly assign a rate for each sender. Thus the volume of bandwidth shared by each legitimate sender is determined by its individual desired throughput, which can be different for

different senders, and the bandwidth utilization efficiency by its transport protocol (*e.g.,* different TCP standards may have different efficiency).

To sum up, FlowPolice creates a dilemma for attackers: To launch DDoS attacks, they have to aggressively inject large volumes of traffic into the network whereas the more traffic they send during the congestion, the less throughput they are allowed. As proved in theoretical analysis (Section 3.6.4) and demonstrated in our evaluation, attackers have to probe the network condition to adjust their rates accordingly. In other words, the optimal strategy for attackers is the least efficient way for launching DDoS attacks. Therefore, by incorporating congestion accountability, FlowPolice can effectively regulate behaviors of all participating flows and mitigate DDoS attacks.

Figure 5.3 and Figure 5.4 are based on the case `(c)` of Figure 5.2, which is the most general and challenging one in our settings. FlowPolice ensures similar throughput for legitimate senders in the other two cases. We omit the results for those two scenarios for clear presentation.

If source spoofing is possible, FlowPolice can still detect misbehaved flows and place rate limitings as long as they are overloading the bottleneck link. FlowPolice terminates the rate limiting only if no packets are lost, which indicates the bottleneck link has enough capacity to handle the arrived packets. Therefore, legitimate users will get their share of the bandwidth. However, the bandwidth allowed for legitimate senders may be smaller than the case where source spoof is eliminated since attackers can generate new flows to keep filling up the bottleneck link. As FlowPolice incorporates the Portcullis [16] protocol so that the number of attack flows is limited, such a problem can be mitigated.

5.2.3 FCT evaluation

To evaluate the FCT performance under attack, we perform a 2MB file transfer between legitimate senders and receivers. We conduct 10 trails for each sender and receiver pair to learn the average FCT. As we can see from the results (Figure 5.5), the average FCT increase is $\sim$1.3$\times$ during attacks. However, we find that without FlowPolice, the file transfer cannot be finished during attacks. Thus FlowPolice mitigates the problem from denial of service to low quality of service. The major cause of FCT inflation is the packet

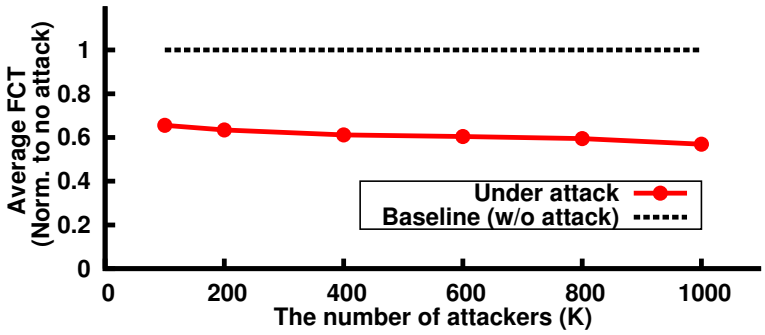


Figure 5.5: Average FCT for legitimate senders.

queuing delay during the initial stage of the flows (as discussed in Section 3.6.3).

CHAPTER 6

DISCUSSION

In this chapter, we make several remarks on the aspects that are not covered so far.

Application layer defense: Note that the design of FlowPolice focuses on solving flooding based DDoS attacks at the network layer. However, FlowPolice can be incorporated with the application layer defense protocols as well. For instance, if an application can detect attack requests and does not return the capabilities carried by attack flows back to attackers, the FlowPolice router will be able to infer large packet losses for attack flows so as to suppress the attack traffic. As a result, attackers will not be able to consume the network resource. In fact, combining with application layer defense protocols provides a new defense primitive for the routers to police flows.

Congestion probe: FlowPolice's overhead can be further reduced if congestion probe is available with the AS. Specially, network operators can probe the network condition via background flows and explicit routing control [32] within its AS, and enable FlowPolice only if some in-network links are congested and flows need to be policed. Therefore, FlowPolice will introduce zero overhead in normal scenarios.

Flow paths: FlowPolice assumes that one flow's path is stable. Although some load balancing protocols, like ECMP, stripe traffic across multiple paths, packets from the same flow are still assigned to the same path. Thus FlowPolice is compatible with ECMP.

CHAPTER 7

RELATED WORK

In this chapter, we discuss related work that has inspired the design of FlowPolice. Generally speaking, we categorize the previous DDoS defense approaches into two major schools (*i.e.,* filtering based approaches and capability based approach), whereas there are other approaches built on different defense primitives.

Filtering-based systems (*e.g.,* IP Traceback [3, 5], AITF[9], Pushback [6, 7], StopIt [10]) try to defend against DDoS attacks by filtering attack flows. Therefore, they need to find a way to differentiate attack flows and legitimate flows. For instance, IP Traceback adopts a packet marking algorithm to construct the path that carries attack traffic so as to block attack flows. AITF aggregates all traffic traversing the same series of ASs as one *Flow* and blocks such flow if the victim AS suspects attacks. Pushback informs upstream routers to block certain type of traffic. StopIt assumes the receiver can detect the attack flows. As mentioned before, filtering based systems often require remote ASs to block attack traffic, which is difficult to enforce. Furthermore, these systems may falsely block legitimate flows since the method used to detect attack flows may have a high false positive rate (*e.g.,* aggregate flows based on traversing ASs).

The capability based systems, such as SIFF [8] and TVA [13], try to suppress attack traffic by only allowing packets carrying valid capabilities, *e.g.,* signatures from the routers on the path. The original design is vulnerable to the aforementioned DoC attack, which is mitigated by the Portcullis [16] protocol. Further, the colluding attackers located on two sides of the victim can grant each other capabilities so that they can still flood the network with privileged packets. NetFence [14] is proposed to achieve per-sender fairness under the colluding scenario.

However, all these approaches assume universal deployment. CRAFT [50] and Mirage [21] are proposed towards real-world deployment. A CRAFT

router emulates TCP states for all traversing flows so that no one can get a greater share than what TCP allows. However, since TCP have many standards and some traffic (*e.g.,* video flows [51]) may even not use standard TCP protocols, CRAFT is not compatible with a real Internet environment and also limits future transport protocol innovation. Mirage [21] is a puzzle based solution and can be incorporated into IPv6 deployment, but it is designed only for securing Web applications.

Other DDoS defense solutions, besides the above two categories, include SpeakUp [17], Phalanx [19], SOS [18] and some future Internet architecture proposals like XIA [52] and SCION [30]. SpeakUp allows legitimate senders to increase their rates to compete with attackers. Such an approach is effective when the bottleneck happens at the application layer so that legitimate users can get more requests processed given all their requests can be delivered. In the case where network is the bottleneck, SpeakUp may potentially congest the network. Phalanx [19] and SOS [18] propose to use large scale overlay networks to defend DDoS attacks. XIA and SCION focus on building the clean-state Internet architecture so as to enhance Internet security, *e.g.,* packet accountability [33].

CHAPTER 8

CONCLUSION

This thesis presents the design, implementation and evaluation of FlowPolice, a new DDoS defense mechanism offering three desirable deployment features. (i) The local deployment feature allows FlowPolice to immediately benefit the first deployed AS without further deployment at other ASs. Such lightweight deployment requirement provides incentives for large scale deployment. (ii) The single deployment feature enables a single FlowPolice router to protect all its downstream bottleneck links that implement a simple prioritization mechanism, *i.e.,* prioritizing FlowPolice's privileged packets. With the flexibility provided by FlowPolice, network operators are able to secure all their ASs with small numbers of deployed routers. (iii) The effectiveness of FlowPolice is merely based on the self-created capability tags, which increases the system robustness. In its design, FlowPolice adopts a congestion feedback mechanism similar to that of NetFence [14], whereas FlowPolice relies on self-created capability feedback to infer remote congestion so that the deploying router can perform flow policing at upstream locations. Our implementation on Linux demonstrates that FlowPolice can scale up to effectively deal with millions of attack flows with small per-packet processing overhead. Our detailed packet-level simulation proves that FlowPolice can effectively mitigate DDoS attacks.

REFERENCES

[1] A. Networks, "Worldwide infrastructure security report, volume IX," http://pages.arbornetworks.com/rs/arbor/images/WISR2014.pdf, 2014.

[2] Prolexic, "Prolexic quarterly global DDoS attack report q2 2014," http://www.prolexic.com/kcresources/attack-report/attack_report_q214/Prolexic-Q22014-Global-Attack-Report-A4.pdf, 2014.

[3] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 295–306, 2000.

[4] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based IP traceback," in *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4. ACM, 2001, pp. 3–14.

[5] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP traceback," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. IEEE, 2001, pp. 878–886.

[6] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, pp. 62–73, 2002.

[7] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against DDoS attacks," *Network and Distributed System Security Symposium*, 2002.

[8] A. Yaar, A. Perrig, and D. Song, "SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on.* IEEE, 2004, pp. 130–143.

[9] K. J. Argyraki and D. R. Cheriton, "Active internet traffic filtering: Real-time response to denial-of-service attacks," in *USENIX Annual Technical Conference, General Track*, 2005, pp. 135–148.

[10] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer dos defense against multimillion-node botnets," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 195–206.

[11] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 365–378.

[12] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing internet denial-of-service with capabilities," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 39–44, 2004.

[13] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 241–252, 2005.

[14] X. Liu, X. Yang, and Y. Xia, "Netfence: preventing internet denial of service from inside out," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 255–266, 2011.

[15] X. Wang and M. K. Reiter, "Mitigating bandwidth-exhaustion attacks using congestion puzzles," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 257–267.

[16] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: protecting connection setup from denial-of-capability attacks," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 289–300, 2007.

[17] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS defense by offense," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 303–314.

[18] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 61–72, 2002.

[19] C. Dixon, T. E. Anderson, and A. Krishnamurthy, "Phalanx: Withstanding multimillion-node botnets," in *NSDI*, vol. 8, 2008, pp. 45–58.

[20] R. Beverly, A. Berger, Y. Hyun, and k claffy, "Understanding the efficacy of deployed internet source address validation filtering," in *Proceedings of the Ninth ACM SIGCOMM/USENIX Internet Measurement Conference (IMC)*, November 2009.

[21] P. Mittal, D. Kim, Y.-C. Hu, and M. Caesar, "Mirage: Towards deployable DDoS defense for web applications," *arXiv preprint arXiv:1110.1060*, 2011.

[22] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2002, pp. 1530–1539.

[23] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants," ser. SIGCOMM '03. New York, USA: ACM, 2003, pp. 75–86.

[24] B. Briscoe, A. Jacquet, C. Di Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe, "Policing congestion response in an internetwork using re-feedback," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 277–288, 2005.

[25] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, 1994.

[26] B. Briscoe, A. Jacquet, T. Moncaster, and A. Smith, "Re-ECN: Adding accountability for causing congestion to TCP/IP," *IETF ID Draft (work in progress)*, 2008.

[27] H. Ballani, P. Francis, and X. Zhang, "A study of prefix hijacking and interception in the internet," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 265–276, 2007.

[28] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 4, pp. 582–592, 2000.

[29] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *Proceedings of ACM SIGCOMM*, 2014.

[30] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "Scion: Scalability, control, and isolation on next-generation networks," in *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 212–227.

[31] W. Simpson, "RFC 1853, IP in IP tunneling," *Network Working Group*, 1995.

[32] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 15–28.

[33] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable internet protocol (AIP)," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 339–350.

[34] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. rfc2827," 2000.

[35] S. Gueron, "Intel advanced encryption standard (AES) instructions set," *White Paper, Intel*, 2010.

[36] "Helion technology, AES cores," http://www.heliontech.com/aes.htm, 2010.

[37] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, August 2013.

[38] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[39] "Alexa top 500 global sites," http://www.alexa.com/topsites, 2014.

[40] S. Savage, "Sting: A TCP-based network measurement tool." in *USENIX Symposium on Internet Technologies and Systems*, vol. 2, 1999, pp. 7–7.

[41] S. Sundaresan, W. De Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband internet performance: a view from the gateway," in *ACM SIGCOMM computer communication review*, vol. 41, no. 4. ACM, 2011, pp. 134–145.

[42] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[43] "NS-3: a discrete-event network simulator," http://www.nsnam.org/.

[44] K. Akdemir, M. Dixon, W. Feghali, P. Fay, V. Gopal, J. Guilford, E. Ozturk, G. Wolrich, and R. Zohar, "Breakthrough AES performance with Intel® AES new instructions," *Intel white paper*, 2010.

[45] "The network simulator: ns-2," http://nsnam.isi.edu/nsnam/index.php/User_Information, 2014.

[46] "Mininet: An instant virtual network on your laptop," http://mininet.org/, 2014.

[47] "The http archive," http://httparchive.org/index.php, 2014.

[48] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[49] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 15–26.

[50] D. Kim, J. T. Chiang, Y.-C. Hu, A. Perrig, and P. Kumar, "Craft: A new secure congestion control architecture," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 705–707.

[51] P. Ameigeiras, J. J. Ramos-Munoz, J. Navarro-Ortiz, and J. M. Lopez-Soler, "Analysis and modelling of youtube traffic," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 4, pp. 360–377, 2012.

[52] D. Naylor, M. K. Mukerjee, P. Agyapong, R. Grandl, R. Kang, M. Machado, S. Brown, C. Doucette, H.-C. Hsiao, D. Han et al., "Xia: architecting a more trustworthy and evolvable internet," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 50–57, 2014.