MiddlePolice: Toward Enforcing Destination-Defined Policies in the Middle of the Internet

Zhuotao Liu[*] Hao Jin[†] Yih-Chun Hu[*] Michael Bailey[*]
[*] University of Illinois at Urbana-Champaign, [†] Nanjing University
[*] {zliu48, yihchun, mdbailey}@illinois.edu, [†] jinhaonju@gmail.com

ABSTRACT

Volumetric attacks, which overwhelm the bandwidth of a destination, are amongst the most common DDoS attacks today. One practical approach to addressing these attacks is to redirect all destination traffic (*e.g.,* via DNS or BGP) to a third-party, DDoS-protection-as-a-service provider (*e.g.,* CloudFlare) that is well provisioned and equipped with filtering mechanisms to remove attack traffic before passing the remaining benign traffic to the destination. An alternative approach is based on the concept of network capabilities, whereby source sending rates are determined by receiver consent, in the form of capabilities enforced by the network. While both third-party scrubbing services and network capabilities can be effective at reducing unwanted traffic at an overwhelmed destination, DDoS-protection-as-a-service solutions outsource all of the scheduling decisions (*e.g.,* fairness, priority and attack identification) to the provider, while capability-based solutions require extensive modifications to existing infrastructure to operate. In this paper we introduce MiddlePolice, which seeks to marry the deployability of DDoS-protection-as-a-service solutions with the destination-based control of network capability systems. We show that by allowing feedback from the destination to the provider, MiddlePolice can effectively enforce destination-chosen policies, while requiring no deployment from unrelated parties.

1. INTRODUCTION

Attacks against availability, such as distributed denial of service attacks (DDoS), continue to plague the Internet. The most common of these attacks, representing roughly 65% of all DDoS attacks last year [41], are volumetric attacks. In these attacks, adversaries seek to deny service by exhausting a victim's network resources and causing congestion. Such attacks are difficult for a victim network to mitigate as the largest of these attacks can exceed the available upstream bandwidth by orders of magnitude. For example, Internet service providers (ISP) reported attacks in excess of 500 Gbps in 2015 [41].

One common solution to this problem is the use of DDoS-protection-as-a-service providers, such as CloudFlare. These providers massively over-provision data centers for peak attack traffic loads and then share this capacity across many customers as needed. When under attack, victims use DNS or BGP to redirect traffic to the provider rather than their own networks. The DDoS-protection-as-a-service provider applies a variety of techniques to scrub this traffic, separating malicious from benign, and then re-injects only the benign traffic back into the network to be carried to the victim. Such methods are appealing, as they require no modification to the existing network infrastructure and can scale to handle very large attacks. However, these cloud-based systems use proprietary attack detection algorithms and filtering which limit the ability of customers to prioritize traffic kinds or choose preferred scheduling policies. Further, existing cloud-based systems assume that all traffic to the victim will be routed first to their infrastructure, an assumption that can be violated by a clever attacker [39, 48].

A second approach to solving volumetric DDoS attacks is network capability-based solutions [9, 12, 13, 35, 42, 43, 51, 52]. Such systems require a source to receive explicit permission before being allowed to contact the destination. Such capabilities are enforced by the network infrastructure itself (*i.e.,* routers) and capabilities range from giving the victim the ability to block traffic from arbitrary sources to giving the victim control over the bandwidth allowed for each flow. A major advantage, then, of these capability-based systems is the ability of the victim to control precisely what and how much traffic it wants to receive. However, these capability-based systems are not without challenges, and most face significant deployment hurdles. For instance, approaches such as TVA [51] and NetFence [35] require secret key management and router upgrades across different Autonomous Systems (ASes). Yet other approaches require clients to modify their network stack to insert customized packet headers, creating additional deployment hurdles.

In this paper, we present MiddlePolice, which seeks to combine the deployability of cloud-based solutions with the destination-based control of capability-based systems. MiddlePolice is built on a set of traffic policing units (referred as mboxes) which rely on a feedback loop of self-generated capabilities to guide scheduling and filtering. MiddlePolice also includes a mechanism to filter nearly all traffic that tries to bypass the mboxes, using only the ACL configuration already present on commodity routers. We implement MiddlePolice as a Linux Kernel Module, and evaluate it extensively over the Internet using cloud infrastructures, on

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10. . . $15.00

DOI: http://dx.doi.org/10.1145/2976749.2978306

our private testbed, and via simulations. Our results show that MiddlePolice can handle large-scale DDoS attacks, and effectively enforce the destination-chosen policies.

2. PROBLEM FORMULATION

2.1 MiddlePolice's Desirable Properties

Readily Deployable and Scalable. MiddlePolice is designed to be readily deployable in the Internet and sufficiently scalable to handle large scale attacks. *To be readily deployable, a system should only require deployment at the destination, and possibly at related parties on commercial terms.* The end-to-end principle of the Internet, combined with large numbers of end points, is what gives rise to its tremendous utility. Because of the diversity of administrative domains, including end points, edge-ASes, and small transit ASes, ASes have varying levels of technological sophistication and cooperativeness. However, some ASes can be expected to help with deployment; many ISPs already provide some sort of DDoS-protection services [2], so we can expect that such providers would be willing to deploy a protocol under commercially reasonable terms. We contrast this with prior capability-based work, which requires deployment at a large number of unrelated ASes in the Internet and client network stack modification, that violates the deployability model.

The goal of being deployable and scalable is the major reason that MiddlePolice is designed to be built into existing cloud-based DDoS defense systems.

Destination-driven Policies. MiddlePolice is designed to provide the destination with fine-grained control over the utilization of their network resources. Throughout the paper, we use "destination" and "victim" interchangeably. Existing cloud-based systems have not provided such functionality. Many previously proposed capability-based systems are likewise designed to work with a single scheduling policy. For instance, CRAFT [29] enforces per-flow fairness, Portcullis [42] and Mirage [38] enforce per-compute fairness, NetFence [35] enforces per-sender fairness, SIBRA [13] enforces per-steady-bandwidth fairness, and SpeakUp [49] enforces per-outbound-bandwidth fairness. If any of these mechanisms is ever deployed, a single policy will be enforced, forcing the victim to accept the choice made by the defense approach. However, no single fairness regime can satisfy all potential victims' requirements. Ideally, MiddlePolice should be able to support victim-chosen policies. In addition to these fairness metrics, MiddlePolice can implement ideas such as ARROW's [43] special pass for critical traffic, and prioritized services for premium clients.

Fixing the Bypass Vulnerability. Existing cloud-based systems rely on DNS or BGP to redirect the destination's traffic to their infrastructures. However, this model opens up the attack of infrastructure bypass. For example, a majority of cloud-protected web servers are subject to IP address exposure [39,48]. Larger victims that SWIP their IP addresses may be unable to keep their IP addresses secret from a determined adversary. In such cases, the adversary can bypass the cloud infrastructures by routing traffic directly to the victims. MiddlePolice includes a readily deployable mechanism to address this vulnerability.

MiddlePolice is designed to augment the existing cloud-based DDoS prevention systems with destination-selectable policies. The literature is replete with capability-based systems that provide a single fairness guarantee with extensive client modification and deployment at non-affiliated ASes. The novelty and challenge of MiddlePolice is therefore architecting a system to move deployment to the cloud while enforcing a wide variety of destination-selectable fairness metrics. Built atop a novel capability feedback mechanism, MiddlePolice meets the challenge, thereby protecting against DDoS more flexibly and deployably.

2.2 Adversary Model and Assumptions

Adversary Model. We consider a strong adversary owning large botnets that can launch strategic attacks and amplify its attack [30]. We assume the adversary is not on-path between any mbox and the victim, since otherwise it could drop all packets. Selecting routes without on-path adversaries is an orthogonal problem and is the subject of active research in next-generation Internet protocols (*e.g.,* SCION [52]).

Well-connected mboxes. MiddlePolice is built on a distributed and replicable set of mboxes that are well-connected to the Internet backbone. We assume the Internet backbone has sufficient capacity and path redundancy to absorb large volumes of traffic, and DDoS attacks against the set of all mboxes can never be successful. This assumption is a standard assumption for cloud-based systems.

Victim Cooperation. MiddlePolice's defense requires the victim's cooperation. If the victim can hide its IP addresses from attackers, it simply needs to remove a MiddlePolice-generated capability carried in each packet and return it back to the mboxes. The victim needs not to modify its layer-7 applications as the capability feedback mechanism is transparent to applications. If attackers can directly send or point traffic (*e.g.,* reflection) to the victim, the victim needs to block the bypassing traffic. MiddlePolice includes a packet filtering mechanism that is immediately deployable on commodity Internet routers.

Cross-traffic Management. We assume that bottlenecks on the path from an mbox to the victim that is shared with other destinations are properly managed, such that cross-traffic targeted at another destination cannot cause unbounded losses of the victim's traffic. Generally, per-destination-AS traffic shaping (*e.g.,* weighted fair share) on these links will meet this requirement.

3. SYSTEM OVERVIEW

MiddlePolice's high-level architecture is illustrated in Figure 1. A MiddlePolice-protected victim redirects its traffic to the mboxes. Each mbox polices traversing traffic to enforce the bandwidth allocation policy chosen by the victim. The traffic policing relies on a feedback loop of MiddlePolice-generated capabilities to eliminate the deployment requirements on downstream paths. When the victim keeps its IP addresses secret, a single deploying mbox can secure the entire path from the mbox to the victim.

For victims whose IP addresses are exposed, attackers can bypass the mboxes and direct attack traffic to the victim. MiddlePolice designs a packet filtering mechanism relying on the ACL on commodity routers or switches to eliminate the traffic that does not traverse any mbox. As long as each bottleneck link is protected by an upstream filter, the bypass attack can be prevented.

4. DETAILED DESIGN OF mboxes

MiddlePolice's traffic policing algorithm (i) probes the

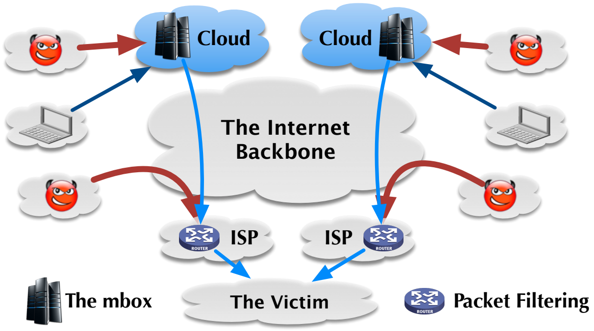


Figure 1. The architecture of MiddlePolice. The mboxes police traffic to enforce victim-chosen policies. The packet filtering discards all traffic bypassing the mboxes.

available downstream bandwidth from each mbox to the victim and (ii) allocates the bandwidth to senders according to the policies chosen by the victim.

Bandwidth Probe. The fundamental challenge of estimating downstream bandwidth is that MiddlePolice requires no deployment at downstream links. Such a challenge is two-sided: an overestimate will cause downstream flooding, rendering traffic policing useless, while an underestimate will waste downstream capacity, reducing performance.

To solve the overestimation problem, MiddlePolice relies on a *capability feedback* mechanism to make senders self-report how many packets they have successfully delivered to the victim. Specifically, upon a packet arrival, the mbox stamps an unforgeable capability in the packet. When the packet is delivered to the victim, MiddlePolice's capability handling module (CHM) deployed on the victim returns the carried capability back to the mbox. If the capability is not returned to the mbox after a sufficiently long time interval (compared with the RTT between the mbox and victim), the mbox will consider the packet lost. Thus, the feedback enables the mbox to infer a packet loss rate (hereinafter, LLR) for each sender. Then the mbox estimates the downstream capacity as the difference between the number of packets received from all senders and packets lost on the downstream path. As the estimation is based on the traffic volume *delivered* to the victim, this approach solves the overestimation problem.

However, the above technique does not overcome the underestimation problem. Specifically, since the traffic demand may be less than downstream capacity, simply using the volume of delivered traffic may cause underestimation. To prevent underestimation, the mbox categorizes packets from each sender as *privileged packets* and *best-effort* packets. Specifically, the mbox maintains a rate window $\mathcal{W}_R$ for each sender to determine the amount of privileged packets allowed for the sender in each period (hereinafter, *detection period*). $\mathcal{W}_R$ is computed based on the above downstream capacity estimation as well as victim-chosen policies. Packets sent beyond $\mathcal{W}_R$ are classified as best-effort packets. The mbox forwards all privileged packets to the victim, whereas the forwarding decisions for best-effort packets are subject to a short-term packet loss rate (hereinafter, SLR). The SLR reflects downstream packet loss rates (congestion) at a RTT granularity. That is, if the downstream is not congested upon an arrival of a best-effort packet, the mbox will forward the packet. Thus, even when the downstream capacity (and thus $\mathcal{W}_R$) is underestimated, the mbox can still further deliver packets as long as the downstream path is not congested.

Fairness Regimes. Each mbox allocates its bandwidth estimate amongst its senders based on the sharing policies chosen by the victim. For policies enforcing global fairness among all senders, all mboxes sharing the same bottleneck share their local observations.

4.1 Information Table

The basis of MiddlePolice's traffic policing is an information table (*iTable*) maintained by each mbox. Each row of the *iTable* corresponds to a single sender. The contents of the *iTable* depend on the victim-selected sharing policy; this section describes *iTable* elements needed for per-sender fairness, and §4.3.5 extends the *iTable* to other fairness regimes. In §6, we describe a mechanism to filter source spoofing at the mbox, so this section ignores source spoofing.

f	$\mathcal{T}_A$	$\mathcal{P}_{id}$	$\mathcal{N}_R$	$\mathcal{N}_D$	$\mathcal{W}_R$	$\mathcal{W}_V$	$\mathcal{L}_R$
64	32	16	32	32	32	128	64

Table 1. Fields of an *iTable* entry and their sizes (bits).

Each sender s_i has one row in the *iTable*, identified by a unique identifier f. The table contents are illustrated in Table 1. Other than f, the remaining fields are updated in each *detection period*. The timestamp $\mathcal{T}_A$ records the current detection period. The capability ID $\mathcal{P}_{id}$ is the maximum number of distinct capabilities generated for s_i. $\mathcal{N}_R$ stores the number of packets received from s_i. $\mathcal{N}_D$ indicates the number of best-effort packets dropped by the mbox. $\mathcal{W}_R$ determines the maximum number of privileged packets allowed for s_i. The verification window $\mathcal{W}_V$ is designed to compute s_i's packet loss rate, whereas $\mathcal{L}_R$ stores the LLR for s_i.

4.2 Capability Computation

For s_i, the mbox generates two types of capabilities: distinct capabilities and common capabilities. The CHM can use either capability to authenticate that the packet has traversed the mbox, though only distinct capabilities are used to infer downstream packet losses.

A distinct capability for s_i is computed as follows:

$$\begin{aligned}\mathcal{C} = IP_{MP} \mid\mid ts \mid\mid \mathcal{P}_{id} \mid\mid f \mid\mid \mathcal{T}_A \mid\mid \\ MAC_{K_s}(IP_{MP} \mid\mid ts \mid\mid \mathcal{P}_{id} \mid\mid f \mid\mid \mathcal{T}_A),\end{aligned} \tag{1}$$

where IP_{MP} is the IP address of the mbox issuing $\mathcal{C}$ and ts is the current timestamp (included to mitigate replay attack). The combination of $\mathcal{P}_{id}||f||\mathcal{T}_A$ ensures the uniqueness of $\mathcal{C}$. The MAC is computed based on a secret key K_s shared by all mboxes. The MAC is 128 bits, so the entire $\mathcal{C}$ consumes ∼300 bits. A common capability is defined as follows

$$\mathcal{C}_c = IP_{MP} \mid\mid ts \mid\mid MAC_{K_s}(IP_{MP} \mid\mid ts). \tag{2}$$

The design of capability incorporates a MAC to ensure that attackers without secure keys cannot generate valid capabilities, preventing capability abuse.

4.3 Traffic Policing Logic

4.3.1 *Populating the* iTable

We first describe how to populate the *iTable*. At time ts, the mbox receives the first packet from s_i. It creates an entry for s_i, with f computed based on s_i's source address, and initializes the remaining fields to zero. It then updates

Symb.	Definition	Value
$\mathcal{D}_p$	The length of the detection period	4s
Th_{cap}	The upper bound of capability ID	128
Th_{rtt}	Maximum waiting time for cap. feedback	1s
Th_{slr}^{drop}	SLR thres. for dropping best-effort pkts	0.05
β	The weight of historical loss rates	0.8
Th_{lpass}	The threshold for calculating LLR	5
$\mathcal{S}_{slr}$	The length limit of the *cTable*	100

Table 2. System parameters.

$\mathcal{T}_A$ to ts, increases both $\mathcal{N}_R$ and $\mathcal{P}_{id}$ by one to reflect the packet arrival and computes a capability using the updated $\mathcal{P}_{id}$ and $\mathcal{T}_A$.

Upon receiving a packet from s_i with arrival time $t_a - \mathcal{T}_A > \mathcal{D}_p$ ($\mathcal{D}_p$ is the length of the detection period), the mbox starts a new detection period for s_i by setting $\mathcal{T}_A = t_a$. The mbox also updates the remaining fields based on the traffic policing algorithm (as described in §4.3.4). The algorithm depends on s_i's LLR and the mbox's SLR, the computation of which is described in the following two sections.

4.3.2 Inferring the LLR for Source s_i

Capability Generation. For each packet from s_i, the mbox generates a distinct capability for the packet if (i) its arrival time $t_a - \mathcal{T}_A < \mathcal{D}_p - Th_{rtt}$, and (ii) the capability ID $\mathcal{P}_{id} < Th_{cap}$. The first constraint ensures that the mbox allows at least Th_{rtt} for each capability to be returned from the CHM. By setting Th_{rtt} well above the RTT from the mbox to the victim, any missing capabilities at the end of the current detection period correspond to lost packets. Table 2 lists the system parameters including Th_{rtt} and their suggested values. MiddlePolice's performance with different settings are studied in §8.3. The second constraint Th_{cap} bounds the number of distinct capabilities issued for s_i in one detection period, and thus bounds the memory requirement. We set $Th_{cap} = 128$ to reduce the LLR sampling error while keeping memory overhead low.

Packets violating either of the two constraints, if any, will carry a common capability (Equation (2)), which is not returned by the CHM or used to learn s_i's LLR. However, it can be used for packet authentication.

Capability Feedback Verification. Let K_{th} denote the number of distinct capabilities the mbox generates for s_i, with capability ID ranging from $[1, K_{th}]$. Each time the mbox receives a returned capability, it checks the capability ID to determine which packet (carrying the received capability) has been received by the CHM. $\mathcal{W}_V$ represents a window with Th_{cap} bits. Initially all the bits are set to zero. When a capability with capability ID i is received, the mbox sets the ith bit in $\mathcal{W}_V$ to one. At the end of the current detection period, the zero bits in the first K_{th} bits of $\mathcal{W}_V$ indicate the losses of the corresponding packets. To avoid feedback reuse, feedback is processed only for capabilities issued in the current period.

LLR Computation. LLR in the kth detection period is computed at the end of the period, *i.e.,* the time when the mbox starts a new detection period for s_i. s_i's lost packets may contain downstream losses and best-effort packets dropped by the mbox ($\mathcal{N}_D$). The number of packets that s_i sent to downstream links is $\mathcal{N}_R - \mathcal{N}_D$, and the downstream packet loss rate is $\frac{V_0}{\mathcal{P}_{id}}$, where V_0 is the number of zero bits in the first $\mathcal{P}_{id}$ bits of $\mathcal{W}_V$. Thus, the estimated number of downstream packet losses is $\mathcal{N}_{loss}^{dstream} = (\mathcal{N}_R - \mathcal{N}_D)\frac{V_0}{\mathcal{P}_{id}}$. Then we have $LLR = (\mathcal{N}_{loss}^{dstream} + \mathcal{N}_D)/\mathcal{N}_R$.

Our strawman design is subject to statistical bias, and may have negative effects on TCP timeouts. In particular, assume one legitimate TCP source recovers from a timeout and sends one packet to probe the network condition. If the packet is dropped again, the source will enter a longer timeout. However, with the strawman design, the source would incorrectly have a 100% loss rate. Adding a low-pass filter can fix this problem: if s_i's $\mathcal{N}_R$ in the current period is less than a small threshold Th_{lpass}, the mbox sets its LLR in the current period to zero. Attackers may exploit this design using on-off attacks [31]. In §4.3.4, we explain how to handle such attacks. Formally, we write LLR as follows

$$LLR = \begin{cases} 0, & \text{if } \mathcal{N}_R < Th_{lpass} \\ \frac{\mathcal{N}_{loss}^{dstream} + \mathcal{N}_D}{\mathcal{N}_R}, & \text{otherwise} \end{cases} \qquad (3)$$

4.3.3 Inferring the SLR

SLR is designed to reflect downstream congestion on a per-RTT basis, and is computed across all flows from the mbox to the victim. Like LLR, SLR is learned through capabilities returned by the CHM. Specifically, the mbox maintains a hash table (*cTable*) to record capabilities used to learn its SLR. The hash key is the capability itself and the value is a single bit value (initialized to zero) to indicate whether the corresponding key (capability) has been returned.

As described in §4.3.2, when a packet arrives (from any source), the mbox stamps a capability on the packet. The capability will be added into *cTable* if it is not a common capability and *cTable*'s length has not reached the predefined threshold $\mathcal{S}_{slr}$. The mbox maintains a timestamp $\mathcal{T}_{slr}$ when the last capability is added into *cTable*. Then, it uses the entire batch of capabilities in *cTable* to learn the SLR. We set $\mathcal{S}_{slr} = 100$ to allow fast capability loading to the *cTable*, while minimizing sampling error from $\mathcal{S}_{slr}$ being too small.

The mbox allows at most Th_{rtt} from $\mathcal{T}_{slr}$ to receive feedback for all capabilities in *cTable*. Some capabilities may be returned before $\mathcal{T}_{slr}$ (*i.e.,* before *cTable* fills). Once a capability in *cTable* is returned, the mbox marks it as received. Upon receiving a new packet with arrival time $t_a > \mathcal{T}_{slr} + Th_{rtt}$, the mbox computes $SLR = \frac{Z_0}{\mathcal{S}_{slr}}$, where Z_0 is the number of *cTable* entries that are not received. The mbox then resets the current *cTable* to be empty to start a new monitoring cycle for SLR.

4.3.4 Traffic Policing Algorithm

We formalize the traffic policing logic in Algorithm 1. Upon receiving a packet P, the mbox retrieves the entry $\mathcal{F}$ in *iTable* matching P (line 8). If no entry matches, the mbox initializes an entry for P.

P is categorized as a privileged or best-effort packet based on $\mathcal{F}$'s $\mathcal{W}_R$ (line 10). All privileged packets are accepted, whereas best-effort packets are accepted conditionally. If P is privileged, the mbox performs necessary capability handling (line 11) before appending P to the privileged queue. The mbox maintains two FIFO queues to serve all *accepted* packets: the privileged queue serving privileged packets and the best-effort queue serving best-effort packets. The privileged queue has strictly higher priority than the best-effort queue at the output port. CapabilityHandling (line 16) executes the capability generation and *cTable* updates (line 37), as detailed in §4.3.2 and §4.3.3.

Algorithm 1: Traffic Policing Algorithm.

```
1  Input:
2      Packet $P$ arrived at time $ts$;
3  Output:
4      $iTable$ updates and possible $cTable$ updates;
5      The forwarding decision of $P$;

6  Main Procedure:
7  begin
8      $\mathcal{F} \leftarrow$ iTableEntryRetrieval($P$);
9      $\mathcal{F}.\mathcal{N}_R \leftarrow \mathcal{F}.\mathcal{N}_R + 1$;
10     if $\mathcal{F}.\mathcal{N}_R < \mathcal{F}.\mathcal{W}_R$ then
           /* Privileged packets */
11         CapabilityHandling($P$, $\mathcal{F}$);
12         Append $P$ to the privileged queue;
13     else
           /* Best-effort packets */
14         BestEffortHandling($P$, $\mathcal{F}$);
       /* Starting a new detection period if necessary */
15     if $ts - \mathcal{F}.\mathcal{T}_A > \mathcal{D}_p$ then iTableHandling($\mathcal{F}$);

16 Function: CapabilityHandling($P$, $\mathcal{F}$):
17 begin
       /* Two constraints for distinct-capability generation */
18     if $\mathcal{F}.\mathcal{P}_{id} < Th_{cap}$ and $ts - \mathcal{F}.\mathcal{T}_A < \mathcal{D}_p - Th_{rtt}$ then
19         $\mathcal{F}.\mathcal{P}_{id} \leftarrow \mathcal{F}.\mathcal{P}_{id} + 1$;
20         Generate capability $\mathcal{C}$ based on Equation (1);
21         cTableHandling($\mathcal{C}$);
22     else
           /* Common capability for packet authentication */
23         Generate capability $\mathcal{C}_c$ based on Equation (2);

24 Function: BestEffortHandling($P$, $\mathcal{F}$):
25 begin
26     if $SLR < Th_{slr}^{drop}$ and $\mathcal{F}.\mathcal{L}_R < Th_{slr}^{drop}$ then
27         CapabilityHandling($P$, $\mathcal{F}$);
28         Append $P$ to the best-effort queue;
29     else
30         Drop $P$;   $\mathcal{F}.\mathcal{N}_D \leftarrow \mathcal{F}.\mathcal{N}_D + 1$;

31 Function: iTableHandling($\mathcal{F}$):
32 begin
33     Compute $recentLoss$ based on Equation (3);
       /* Consider the historical loss rate */
34     $\mathcal{F}.\mathcal{L}_R \leftarrow (1 - \beta) \cdot recentLoss + \beta \cdot \mathcal{F}.\mathcal{L}_R$;
35     $\mathcal{W}_R \leftarrow$ BandwidthAllocationPolicy($\mathcal{F}$);
36     Reset $\mathcal{W}_V$, $\mathcal{P}_{id}$, $\mathcal{N}_R$ and $\mathcal{N}_D$ to zero;

37 Function: cTableHandling($\mathcal{C}$):
38 begin
       /* One batch of cTable is not ready */
39     if $cTable.length < \mathcal{S}_{slr}$ then
40         Add $\mathcal{C}$ into $cTable$;
41     if $cTable.length == \mathcal{S}_{slr}$ then $\mathcal{T}_{slr} \leftarrow ts$ ;
```

If P is a best-effort packet, its forwarding decision is subject to the mbox's SLR and $\mathcal{F}'s$ LLR (line 24). If the SLR exceeds Th_{slr}^{drop}, indicating downstream congestion, the mbox discards P. Further, if $\mathcal{F}$'s LLR is already above Th_{slr}^{drop}, the mbox will not deliver best-effort traffic for $\mathcal{F}$ as well since $\mathcal{F}$ already experiences severe losses. Th_{slr}^{drop} is set to be few times larger than a TCP flow's loss rate in normal network condition [47] to absorb burst losses. If the mbox decides to accept P, it performs capability handling (line 27).

Finally, if P's arrival triggers a new detection period for $\mathcal{F}$ (line 15), the mbox performs corresponding updates for $\mathcal{F}$ (line 31). To determine $\mathcal{F}$'s LLR, the mbox incorporates both the recent LLR (*recentLoss*) obtained in the current detection period based on Equation (3) and $\mathcal{F}$'s historical loss rate $\mathcal{L}_R$. This design prevents attackers from hiding their previous packet losses via on-off attacks. $\mathcal{F}$'s $\mathcal{W}_R$ is updated based on the victim-selected policy (line 35), as described below.

4.3.5 Bandwidth Allocation Policies

We list the following representative policies that may be chosen to implement in BandwidthAllocationPolicy.

NaturalShare: for each sender, the mbox sets its $\mathcal{W}_R$ for the next period to the number of delivered packets from the sender in the current period. The design rationale is that the mbox allows a rate that the sender can sustainably transmit without experiencing a large LLR.

PerSenderFairshare allows the victim to enforce per-sender fair share at bottlenecks. Each mbox fairly allocates its estimated total downstream bandwidth to the senders that reach the victim through the mbox. To this end, the mbox maintains the total downstream bandwidth estimate $\mathcal{N}_{size}^{total}$, which it allocates equally among all senders.

To ensure global fairness among all senders, two mboxes sharing the same bottleneck (*i.e.,* the two paths connecting the two mboxes with the victim both traverse the bottleneck link) share their local observations. We design a co-bottleneck detection mechanism using SLR correlation: if two mboxes' observed SLRs are correlated, they share a bottleneck with high probability. In §8.3, we evaluate the effectiveness of this mechanism.

PerASFairshare is similar to PerSenderFairshare except that the mbox fairly allocates $\mathcal{N}_{size}^{total}$ on a per-AS basis. This policy mimics SIBRA [13], preventing bot-infested ASes from taking bandwidth away from legitimate ASes.

PerASPerSenderFairshare is a hierarchical fairness regime: the mbox first allocates $\mathcal{N}_{size}^{total}$ on a per-AS basis, and then fairly assigns the share obtained by each AS among the senders of the AS.

PremiumClientSupport provides premium service to premium clients, such as bandwidth reservation for upgraded ASes. The victim pre-identifies its premium clients to MiddlePolice. PremiumClientSupport can be implemented together with the aforementioned allocation policies.

5. PACKET FILTERING

When the victim's IP addresses are kept secret, attackers cannot bypass MiddlePolice's upstream mboxes to route attack traffic directly to the victim. In this case, the downstream packet filtering is *unnecessary* since MiddlePolice can throttle attacks at the upstream mboxes. However, in case of IP address exposure [39, 48], the victim needs to deploy a packet filter to discard bypassing traffic. MiddlePolice designs a filtering mechanism that extends to commodity routers the port-based filtering of previous work [21, 22]. Unlike prior work, the filtering can be deployed upstream of the victim as a commercial service.

5.1 Filtering Primitives

Although the MAC-incorporated capability can prove that a packet indeed traverses an mbox, it requires upgrades from deployed commodity routers to perform MAC computation to filter bypassing packets. Thus, we invent a mechanism based on the existing ACL configurations on commodity routers. Specifically, each mbox encapsulates its traversing packets into UDP packets (similar techniques have been applied in VXLAN and [25]), and uses the UDP source and destination ports (a total of 32 bits) to carry an authenticator, which is a shared secret between the mbox and the filtering point. As a result, a 500Gbps attack (the largest attack viewed by Arbor Networks [41]) that uses random port numbers will be reduced to ∼100bps since the chance of a correct guess is 2^{-32}. The shared secret can be negotiated periodically based on a cryptographically secure pseudo-random number generator. We do not rely on UDP source address for filtering to avoid source spoofing.

5.2 Packet Filtering Points

Deployed filtering points should have sufficient bandwidth so that the bypassing attack traffic cannot cause packet losses prior to the filtering. The filtering mechanism should be deployed at, or upstream of, each bottleneck link caused by the DDoS attacks. For instance, for a victim with high-bandwidth connectivity, if the bottleneck link is an internal link inside the victim's network, the victim can deploy the filter at the inbound points of its network. If the bottleneck link is the link connecting the victim with its ISP, the victim can can work with its ISP, on commercially reasonable terms, to deploy the filter deeper in the ISP's network such that the bypassing traffic cannot reach the victim's network. Working with the ISPs does not violate the deployment model in §2 as MiddlePolice never requires deployment at unrelated ASes.

6. SOURCE VALIDATION

MiddlePolice punishes senders even after an offending flow ends. Such persistence can be built on source authentication or any mechanism that maintains sender accountability across flows. As a proof-of-concept, we design a source validation mechanism that is specific to HTTP/HTTPS traffic. The mechanism ensures that a sender is on-path to its claimed source IP address. This source verifier is completely transparent to clients.

Our key insight is that the HTTP Host header is in the first few packets of each connection. As a result, the mbox monitors a TCP connection and reads the Host header. If the Host header reflects a generic (not sender-specific) hostname (*e.g.,* victim.com), the mbox intercepts this flow, and redirects the connection (HTTP 302) to a Host containing a token cryptographically generated from the sender's claimed source address, *e.g.,* T.victim.com, where T is the token. If the sender is on-path, it will receive the redirection, and its further connection will use the sender-specific hostname in the Host header. When an mbox receives a request with a sender-specific Host, it verifies that the Host is proper for the claimed IP source address (if not, the mbox initiates a new redirection), and forwards the request to the victim. Thus, by performing source verification entirely at the mbox, packets from spoofed sources cannot consume any downstream bandwidth from the mbox to the victim.

If the cloud provider hosting the mbox is trusted (for instance, large CDNs have CAs trusted by all major browsers), the victim can share its key such that HTTPS traffic can be handled in the same way as HTTP traffic. For untrusted providers [22, 32], the mbox relays the encrypted connection to the victim, which performs Host-header-related operations. The victim terminates unverified senders without returning capabilities to the mbox, so the additional traffic from the unverified senders is best-effort under Algorithm 1. In this case, packets from spoofed sources consume limited downstream bandwidth but do not rely on the trustworthiness of the cloud provider. We acknowledge that performing source validation at the victim is subject to the DoC attack [11] in which attackers flood the victim with new connections to slow down the connection-setup for legitimate clients. This attack is mitigated if the source validation is completely handled by the mbox.

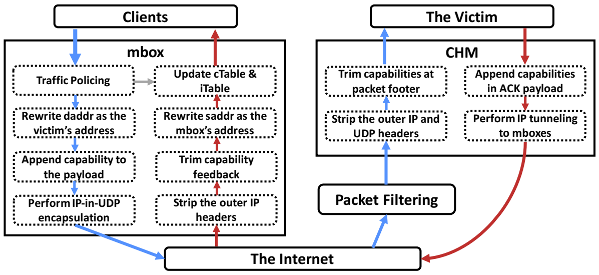

Clients
mbox
Traffic Policing
Update cTable & iTable
Rewrite daddr as the victim's address
Rewrite saddr as the mbox's address
Append capability to the payload
Trim capability feedback
Perform IP-in-UDP encapsulation
Strip the outer IP headers
The Victim
CHM
Trim capabilities at packet footer
Append capabilities in ACK payload
Strip the outer IP and UDP headers
Perform IP tunneling to mboxes
Packet Filtering
The Internet

Figure 2. The software stack of the mbox and CHM.

7. IMPLEMENTATION

We have a full implementation of MiddlePolice.

7.1 The Implementation of CHM and mboxes

The mboxes and the CHM at the victim are implemented based on the NetFilter Linux Kernel Module, which combined have ∼1500 lines of C code (excluding the capability generation code). The software stack of our implementation is illustrated in Figure 2.

All inbound traffic from clients to an mbox is subject to the traffic policing whereas only accepted packets go through the capability-related processing. Packet dropping due to traffic policing triggers *iTable* updates. For each accepted packet, the mbox rewrites its destination address as the victim's address to point the packet to the victim. To carry capabilities, rather than defining a new packet header, the mbox appends the capabilities to the end of the original data payload, which avoids compatibility problems at intermediate routers and switches. The CHM is responsible for trimming these capabilities to deliver the original payload to the victim's applications. If the packet filter is deployed, the mbox performs the IP-in-UDP encapsulation, and uses the UDP source and destination port number to carry an authenticator. All checksums need to be recomputed after packet manipulation to ensure correctness. ECN and encapsulation interactions are addressed in [14].

To avoid packet fragmentation due to the additional 68 bytes added by the mbox (20 bytes for outer IP header, 8 bytes for the outer UDP header, and 40 bytes reserved for a capability), the mbox needs to be *a priori* aware of the MTU M_d on its path to the victim. Then the mbox sets its MSS to no more than $M_d - 68 - 40$ (the MSS is 40 less than the MTU). We do not directly set the MTU of the mbox's NIC to rely on the path MTU discovery to find the right packet size because some ISPs may block ICMP packets. On our testbed, $M_d = 1500$, so we set the mbox's MSS to 1360.

Upon receiving packets from upstream mboxes, the CHM strips their outer IP/UDP headers and trims the capabilities. To return these capabilities, the CHM piggybacks capabilities to the payload of ACK packets. To ensure that a capability is returned to the mbox issuing the capability even if the Internet path is asymmetric, the CHM performs IP-in-IP encapsulation to tunnel the ACK packets to the right mbox. We allow one ACK packet to carry multiple capabilities since the victim may generate cumulative ACKs rather than per-packet ACKs. Further, the CHM tries to pack more capabilities in one ACK packet to reduce the capability feedback latency at the CHM. The number of capabilities carried in one ACK packet is stored in the TCP option (the 4-bit `res1` option). Thus, the CHM can append up to 15 capabilities in one ACK packet if the packet has enough space and the CHM has buffered enough capabilities.

Upon receiving an ACK packet from the CHM, the mbox strips the outer IP header and trims the capability feedback (if any) at the packet footer. Further, the mbox needs to rewrite the ACK packet's source address back to its own address, since the client's TCP connection is expecting to communicate with the mbox. Based on the received capability feedback, the mbox updates the *iTable* and *cTable* accordingly to support the traffic policing algorithm.

7.2 Capability Generation

We use the AES-128 based CBC-MAC, based on the Intel AES-NI library, to compute MACs, due to its fast speed and availability in modern CPUs [6, 24]. We port the capability implementation (∼400 lines of C code) into the mbox and CHM kernel module. The mbox needs to perform both capability generation and verification whereas the CHM performs only verification.

8. EVALUATION

8.1 The Internet Experiments

This section studies the path length and latency inflation for rerouting clients' traffic to mboxes hosted in the cloud.

8.1.1 Path Inflation

We construct the AS level Internet topology based on the CAIDA AS relationships dataset [4], including 52680 ASes and their business relationships [17]. To construct the communication route, two constraints are applied based on the routes export policies in [20, 23]. First, an AS prefers customer links over peer links and peer links over provider links. Second, a path is valid only if each AS providing transit is paid. Among all valid paths, an AS prefers the path with least AS hops (a random tie breaker is applied if necessary). As an example, we use Amazon EC2 as the cloud provider to host the mboxes, and obtain its AS number based on the report [5]. Amazon claims 11 ASes in the report. We first exclude the ASes not appearing in the global routing table, and find that AS 16509 is the provider for the remaining Amazon ASes, so we use AS 16509 to represent Amazon.

We randomly pick 2000 ASes as victims, and for each victim we randomly pick 1500 access ASes. Among all victims, 1000 victims are stub ASes without direct customers and the remaining victims are non-stub ASes. For each AS-victim pair, we obtain the direct route from the access AS to the victim, and the rerouted path through an mbox. Table 3 summarizes the route comparison. N^{hop}_{infla} is the average AS-hop inflation of the rerouted path compared with the direct route. P^{short}_{cut} is the percentage of access ASes that can reach the victim with fewer hops after rerouting and P^{no}_{infla} is percentage of ASes without hop inflation.

Victims	N^{hop}_{infla}	P^{short}_{cut}	P^{no}_{infla}
Non-stub ASes	1.1	10.6%	22.2%
Stub ASes	1.5	8.4%	18.0%
Overall	1.3	9.5%	20.1%

Table 3. Rerouting traffic to mboxes causes small AS-hop inflation, and ∼10% access ASes can even reach the victim with fewer hops through mboxes.

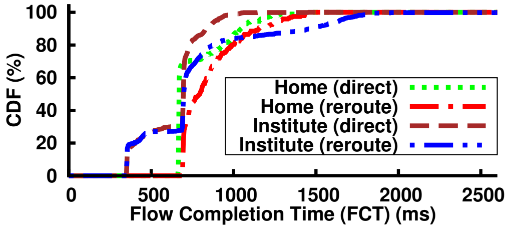


Figure 3. [Internet] FCTs for direct paths and rerouted paths under various Internet conditions.

Overall, it takes an access AS 1.3 more AS-hops to reach the victim after rerouting. Even for stub victims, which are closer the Internet edge, the average hop inflation is only 1.5. We also notice that ∼10% ASes have *shorter* paths due to the rerouting.

Besides EC2, we also perform path inflation analysis when mboxes are hosted by CloudFlare. The results show that the average path inflation is about 2.3 AS-hops. For any cloud provider, MiddlePolice has the *same path inflation* as the cloud-based DDoS solutions hosted by the same cloud provider, since capability feedback is carried in ACK packets. As such, deploying MiddlePolice into existing cloud-based systems does not increase path inflation.

8.1.2 Latency Inflation

In this section, we study the latency inflation caused by the rerouting. In our prototype running on the Internet, we deploy 3 mboxes on Amazon EC2 (located in North America, Asia and Europe), one victim server in a US university and about one hundred senders (located in North America, Asia and Europe) on PlanetLab [1] nodes. We also deploy few clients on personal computers to test MiddlePolice in home network. The wide distribution of clients allows us to evaluate MiddlePolice on various Internet links. We did not launch DDoS attacks over the Internet, which raises ethical and legal concerns. Instead, we evaluate how MiddlePolice may affect the clients in the normal Internet without attacks, and perform the experiments involving large scale DDoS attacks on our private testbed and in simulation.

In the experiment, each client posts a 100KB file to the server, and its traffic is rerouted to the nearest mbox before reaching the server. We repeat the posting on each client 10,000 times to reduce sampling error. We also run the experiment during both peak hours and midnight (based on the server's timezone) to test various network conditions. As a control, clients post the files to server via direct paths.

Figure 3 shows the CDF of the flow completion times (FCTs) for the file posting. Overall, we notice ∼9% average FCT inflation, and less than 5% latency inflation in home network. Therefore, traffic rerouting introduces small extra latency to the clients.

MiddlePolice's latency inflation includes both rerouting-induced networking latency and capability-induced compu-

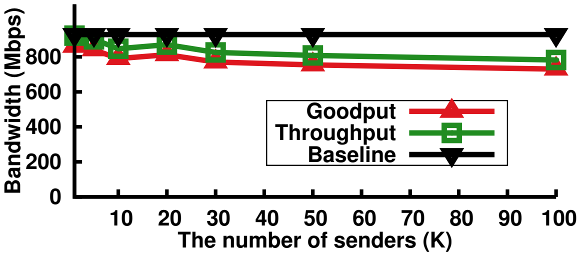


Figure 4. [Testbed] Throughput and goodput when MiddlePolice policies different numbers of senders.

tational overhead. As evaluated in §8.2.1, the per-packet processing latency overhead caused by capability computation is ∼1.4 μs, which is negligible compared with typical Internet RTTs. Thus, MiddlePolice has latency almost identical to the existing cloud-based DDoS mitigation.

8.2 Testbed Experiments

8.2.1 *Traffic Policing Overhead*

In this section, we evaluate the traffic policing overhead on our testbed. We organize three servers as one sender, one mbox and one receiver. All servers, shipped with a quad-core Intel 2.8GHz CPU, run the 3.13.0 Linux kernel. The mbox is installed with multiple Gigabit NICs to connect both the sender and receiver. A long TCP flow is established between the sender and receiver, via the mbox, to measure the throughput. To emulate a large number of sources, the mbox creates an *iTable* with N entries. Each packet from the sender triggers a table look up for a random entry. We implement a two-level hash table in the kernel space to reduce the look up latency. Then the mbox generates a capability based on the obtained entry.

Figure 4 shows the measured throughput and goodput under various N. The goodput is computed by subtracting the additional header and capability size from the total packet size. The baseline throughput is obtained without MiddlePolice. Overall, the policing overhead in high speed network is small. When a single mbox deals with 100,000 sources sending *simultaneously*, throughput drops by ∼10%. Equivalently, MiddlePolice adds around 1.4 microseconds latency to each packet processed. By replicating mboxes, the victim can distribute the workload across many mboxes when facing large scale attacks.

8.2.2 *Enforce Destination-Defined Policies*

We now evaluate MiddlePolice's performance for enforcing victim-defined policies, along with the effectiveness of filtering bypassing traffic. This section evaluates *pure* MiddlePolice. In reality, once built into cloud-based systems, MiddlePolice needs only to process traffic that passes their pre-deployed defense.

Testbed Topology. Figure 5 illustrates the network topology, including a single-homed victim AS purchasing 1Gbps bandwidth from its ISP, an mbox and 10 access ASes. The ISP is emulated by a Pronto-3297 48-port Gigabit switch to support packet filtering. The mbox is deployed on a server with multiple Gigabit NICs, and each access AS is deployed on a server with a single NIC. We add 100ms latency at the victim via Linux traffic control to emulate the typical Internet RTT. To emulate large scale attacks, 9 ASes are compromised. Attackers adopt a hybrid attack profile: 6 attack ASes directly send large volumes of traffic to the victim, emulating amplification-based attacks, and the remaining attack ASes route traffic through the mbox. Thus, the total volume of attack traffic is 9 times as much as the victim's bottleneck link capacity. Both the inbound and outbound points of the mbox are provisioned with 4Gbps bandwidth to ensure the mbox is not the bottleneck, emulating that the mbox is hosted in the cloud.

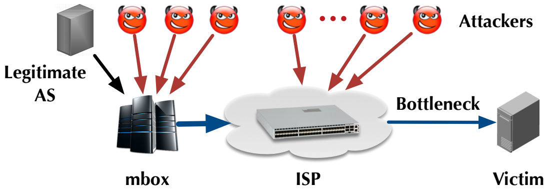


Figure 5. Testbed network topology.

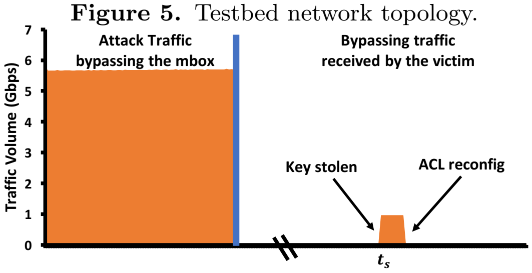
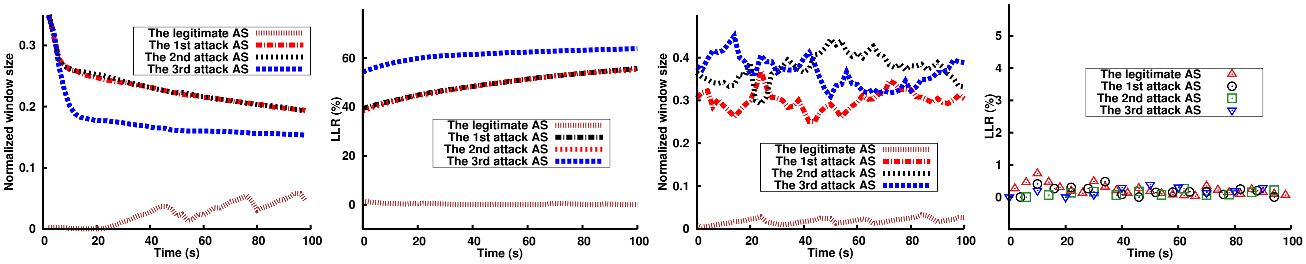


Figure 6. [Testbed] Packet filtering via ACL.

Packet Filtering. We first show the effectiveness of the packet filter. Six attack ASes spoof the mbox's source address and send 6Gbps UDP traffic to the victim. The attack ASes scan all possible UDP port numbers to guess the shared secret. Figure 6 shows the volume of attack traffic bypassing the mbox and its volume received by the victim. As the chance of a correct guess is very small, the filter can effectively stop the bypassing traffic from reaching the victim. Further, even if the shared secret were stolen by attackers at time t_s, the CHM would suddenly receive large numbers of packets without valid capabilities. Since packets traversing the mbox carry capabilities, the CHM realizes that the upstream filtering has been compromised. The victim then re-configures the ACL using a new secret to recover from key compromise. The ACL is effective within few milliseconds after reconfiguration. Thus, the packet filtering mechanism can promptly react to a compromised secret.

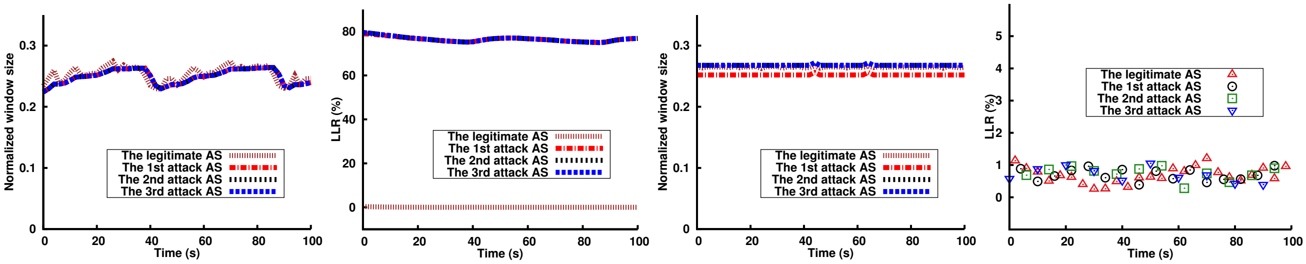
NaturalShare and PerASFairshare Policies. In this section, we first show that the mbox can enforce the NaturalShare and PerASFairshare policies. We use the default parameter setting in Table 2, and defer detailed parameter study in §8.3. Since MiddlePolice conditionally allows an AS to send faster than its $\mathcal{W}_R$, we use the *window size*, defined as the larger value between an AS's $\mathcal{W}_R$ and its delivered packets to the victim, as the performance metric. For clear presentation, we normalize the window size to the maximum number of 1.5KB packets deliverable through a 1Gbps link in one detection period. We do not translate window sizes to throughput because packet sizes vary.

Attackers adopt two representative strategies: (i) they send flat rates regardless of packet losses, and (ii) they dynamically adjust their rates based on packet losses (reactive attacks). To launch flat-rate attacks, the attackers keep sending UDP traffic to the victim. The CHM uses a dedi-



(a) Window sizes in flat-attacks. (b) LLRs in flat-attacks. (c) Window sizes in reactive att. (d) LLRs in reactive attacks.

Figure 7. [Testbed] Enforcing the NaturalShare policy. The legitimate AS gradually obtains a certain amount of bandwidth under flat-rate attacks since attackers' window sizes drop consistently over time (Figure 7(a)) due to their high LLRs (Figure 7(b)). However, the attack ASes can consume over 95% of the bottleneck bandwidth via reactive attacks (Figure 7(c)) while maintaining low LLRs similar to the legitimate AS's LLR (Figure 7(d)).



(a) Window sizes in flat-attacks. (b) LLRs in flat-attacks. (c) Window sizes in reactive att. (d) LLRs in reactive attacks.

Figure 8. [Testbed] Enforcing the PerASFairshare policy. The legitimate AS can obtain at least the per-AS fair rate at the bottleneck regardless of the attack strategies (Figures 8(a) and 8(c)). Further, the legitimate AS gains slightly more bandwidth than the attackers under flat-rate attacks as the attack ASes have large LLRs (Figure 8(b)).

cated flow to return received capabilities to the mbox since no ACK packets are generated for UDP traffic. One way of launching reactive attacks is that the attackers simultaneously maintain many more TCP flows than the legitimate AS. Such a many-to-one communication pattern allows the attackers to occupy almost the entire bottleneck, even through each single flow seems completely "legitimate".

The legitimate AS always communicates with the victim via a long-lived TCP connection.

Figure 7 shows the results for the NaturalShare policy. As the bottleneck is flooded by attack traffic, the legitimate AS is forced to enter timeout at the beginning, as illustrated in Figure 7(a). The attackers' window sizes are decreasing over time, which can be explained via Figure 7(b). As the volume of attack traffic is well above the bottleneck's capacity, all attack ASes' LLRs are well above Th_{slr}^{drop}. Thus, the mbox drops all their best-effort packets. As a result, when one attack AS's window size is $W(t)$ in detection period t, then $W(t+1) \leq W(t)$ since in period $t+1$ any packet sent beyond $W(t)$ is dropped. Further, any new packet losses from the attack AS, caused by an overflow at the bottleneck buffer, will further reduce $W(t+1)$. Therefore, all attack ASes' window sizes are consistently decreasing over time, creating spare bandwidth at the bottleneck for the legitimate AS. As showed in Figure 7(a), the legitimate AS gradually recovers from timeouts.

The NaturalShare policy, however, cannot well protect the legitimate AS if the attackers adopt the reactive attack strategy. By adjusting the sending rates based on packet losses, the attack ASes can keep their LLRs low enough to regain the advantage of delivering best-effort packets. Meanwhile, they can gain much more bandwidth by initiating more TCP flows. Figure 7(c) shows the window sizes when each attack AS starts 200 TCP flows whereas the legitimate AS has only one. The attackers consume over 95% of the bottleneck bandwidth, while keeping low LLRs similar to that of the legitimate AS (Figure 7(d)).

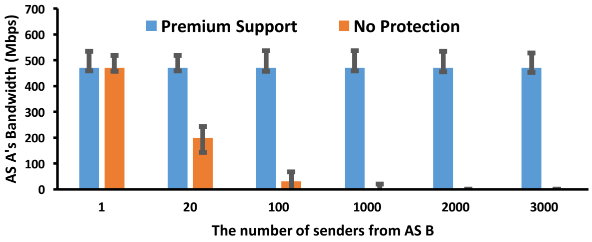
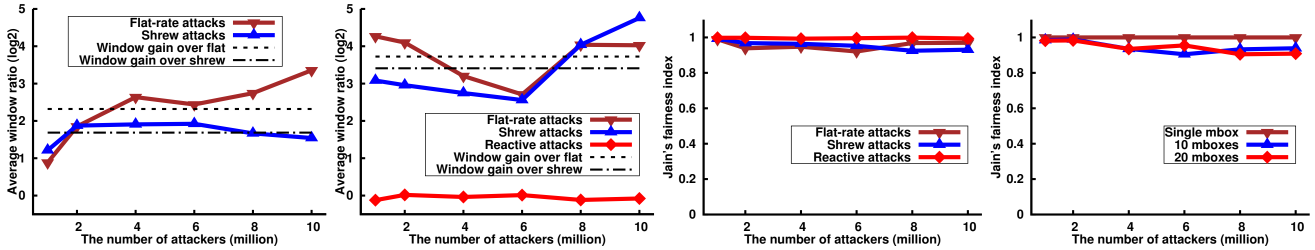


Figure 9. [Testbed] MiddlePolice ensures that the premium client (AS A) receives consistent bandwidth.

Figure 8 shows the results for the PerASFairshare policy. Figures 8(a) and 8(c) demonstrate that the legitimate AS receives at least per-AS fair rate at the bottleneck regardless of the attack strategies, overcoming the shortcomings of the NaturalShare policy. Further, under flat-rate attacks, the legitimate AS has slightly larger window sizes than the attackers since, again, the mbox does not accept any best-effort packets from the attackers due to their high LLRs (as showed in Figure 8(b)).

PremiumClientSupport Policy. This section evaluates the PremiumClientSupport policy. We consider a legitimate AS (AS A) that is a premium client which reserves half of the bottleneck bandwidth. Figure 9 plots AS A's bandwidth



(a) NaturalShare. (b) PerSenderFairshare. (c) Jain's fairness index (FI). (d) FI for various mbox counts.

Figure 10. [Simulation] Evaluating NaturalShare & PerSenderFairshare in large scale. Figures 10(a) and 10(b) show that the clients' average window size is larger than that of the attackers under both flat-rate and shrew attacks. Figure 10(c) proves that the clients' window sizes converge to fairness in the PerSenderFairshare policy. Figure 10(d) shows that MiddlePolice can enforce strong fairness among all senders even without coordination among the mboxes.

when the number of senders from the attack ASes increases. With the PremiumClientSupport policy, MiddlePolice ensures AS A receives consistent bandwidth regardless of the number of senders from the attack ASes. However, without such a policy, the attack ASes can selfishly take away the majority of bottleneck bandwidth by involving more senders.

8.3 Large Scale Evaluation

In this section, we further evaluate MiddlePolice via large scale simulations on ns-3 [3]. We desire to emulate real-world DDoS attacks in which up to millions of bots flood a victim. To circumvent the scalability problem of ns-3 at such a scale, we adopt the same approach in NetFence [35], *i.e.,* by fixing the number of nodes (∼5000) and scaling down the link capacity proportionally, we can simulate attack scenarios where 1 million to 10 million attackers flood a 40Gbps link. The simulation topology is similar to the testbed topology, except that all attackers are connected to the mbox.

Besides the flat-rate attacks and reactive attacks, we also consider the on-off shrew attacks [31] in the simulations. Both the on-period and off-period in shrew attacks are 1s. The number of attackers is 10 times larger than that of legitimate clients. In flat-rate attacks and shrew attacks, the attack traffic volume is 3 times larger than the capacity of the bottleneck. In reactive attacks, each attacker opens 10 connections, whereas a client has one. The bottleneck router buffer size is determined based on [10], and the RTT is 100ms.

NaturalShare & PerSenderFairshare in Scale. Figure 10 shows the results for enforcing NaturalShare and PerSender-Fairshare policies with default parameter settings. We plot the ratio of clients' average window size to attackers' average window size for the NaturalShare policy in Figure 10(a). For flat-rate attacks and shrew attacks, it may be surprising that the clients' average window size is larger than that of the attackers. Detailed trace analysis shows that it is because that the window sizes of a large portion of attackers keep decreasing, as we explained in our testbed experiment. As the number of attackers is much larger than the client count, the attackers' average window size turns out to be smaller than that of the clients, although the absolute volume of attack traffic may be still higher. Under reactive attacks, the clients' average window size (almost zero) is too small to be plotted in Figure 10(a).

Figure 10(b) shows that the clients enjoy even larger window ratio gains under the PerSenderFairshare policy in flat-rate and shrew attacks because even more attackers enter the window dropping mode. Further, the PerSenderFairshare ensures that the clients' average window size is close to the per-sender fair rate in reactive attacks. Figure 10(c) demonstrates that each client's window size converges to per-client fairness as Jain's fairness index [16] (FI) is close to 1.

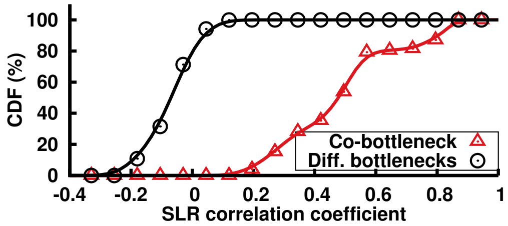


Figure 11. [Simulation] The SLR correlation coefficient reflects whether two mboxes share a bottleneck.

mbox Coordination. To enforce global per-sender fairness, the mboxes sharing the same bottleneck link share their local observations (§4.3.5). We first investigate how bad the FI can be without such inter-mbox coordination. We reconstruct the topology to create multiple mboxes, and map each client to a random mbox. The attackers launch reactive attacks. The results, plotted in Figure 10(d), show that the FI drops slightly, by ∼8%, even if 20 mboxes make local rate allocations without any coordination among them.

To complete our design, we further propose the following co-bottleneck detection mechanism. The design rationale is that if two mboxes' SLR observations are correlated, they share a bottleneck with high probability. To validate this, we rebuild the network topology to create the scenarios where two mboxes share and do not share a bottleneck, and study the correlation coefficient of their SLRs. We compute one coefficient for every 100 SLR measurements from each mbox. Figure 11 shows the CDF of the coefficient. Clearly, the coefficient reflects whether the two mboxes share a bottleneck. Thus, by continuously observing such correlation between two mboxes' SLRs, MiddlePolice can determine with increasing certainty whether or not they share a bottleneck, and can configure their coordination accordingly.

Parameter Study. We evaluate MiddlePolice using different parameters than the default values in Table 2. We mainly focus on $\mathcal{D}_p$, Th^{drop}_{slr} and β. For each parameter, we vary its value to obtain the clients' average window size under the 10-million bot attack. The results showed in Table 5 are normalized to the window sizes obtained using the default parameters in Table 2.

	Pushback [37]	SIFF [50], TVA [51]	Netfence [35]	Phalanx [18]	Mirage [38]	SIBRA [13]	MiddlePolice
Source upgrades	**No**	Yes	Yes	Yes	Yes	Yes	**No**
Dest. upgrades	**No**	Yes	Yes	Yes	Yes	Yes	Yes
AS deployment	Unrelated	Unrelated	Unrelated	Unrelated	**Related**	Unrelated	**Related**
Router support	$O(N)$ states	Cryptography; $O(N)$ states for [51]	$O(N)$ states; Cryptography	$O(N)$ states	Larger memory	**None**	**None**
Fairness regimes	None	None	Per-sender	None	Per-compute	Per-AS	**Victim-chosen**
Other requirements	**None**	New header	New header; Passport [33]	New header	Puzzle; IPv6 upgrade	Redesign the Internet	**None**

Table 4. Property comparison with other research proposals. "$O(N)$ states" means that the number of states maintained by a router increases with the number of attackers. "Cryptography" means that a router needs to support cryptography operation, *e.g.,* MAC computation. "Puzzle" means that the mechanism requires computational puzzle distribution.

(a) The NaturalShare Policy

	$\mathcal{D}_p$		Th_{slr}^{drop}		β	
	2s	8s	0.03	0.1	0.5	0.9
Flat	1.1	0.17	0.78	0.39	1.1	0.78
Shrew	1.3	0.65	0.77	1.0	1.2	0.80

(b) The PerSenderFairshare Policy

	$\mathcal{D}_p$		Th_{slr}^{drop}		β	
	2s	8s	0.03	0.1	0.5	0.9
Flat	1.0	1.1	1.0	0.69	0.85	0.81
Shrew	1.1	0.98	0.72	0.83	1.0	0.98
Reactive	1.0	0.99	1.0	0.94	1.0	1.0

Table 5. [Simulation] Clients' average window size under different parameter settings.

Under the NaturalShare policy, the shorter $\mathcal{D}_p$ produces a larger window size for legitimate clients since each sender's $\mathcal{W}_R$ is updated per-period so that a smaller $\mathcal{D}_p$ causes faster cut in attackers' window sizes. For Th_{slr}^{drop}, a smaller value slows down the clients' recovery whereas a larger value allows larger window sizes for attackers. Both will reduce the clients' share. A larger β has negative effects as it takes more time for the clients to recover to a low LLR.

With the PerSenderFairshare policy, MiddlePolice's performance is more consistent under different parameter settings. The most sensitive parameter is Th_{slr}^{drop} because it determines whether one source can send best-effort traffic.

9. RELATED WORK

In this section, we briefly discuss previous academic work. Previous research approaches can be generally categorized into capability-based approaches (SIFF [50], TVA [51], NetFence [35]), filtering-based approaches (Traceback [44, 46], AITF [12], Pushback [26,37], StopIt [34]), overlay-based approaches (Phalanx [18], SOS [28], Mayday [7]), deployment-friendly approaches (Mirage [38], CRAFT [29]), approaches based on new Internet architectures (SCION [52], SIBRA [13], XIA [40], AIP [8]), and others (SpeakUp [49], SDN-based [19, 45], CDN-based [22]). We summarize the properties of one or two approaches from each category in Table 4. The comparison shows that MiddlePolice requires the *least* deployment (no source upgrades, no additional router support and no deployment from unrelated ASes) while providing the *strongest* property (enforcing destination-chosen policies).

10. DISCUSSION

We briefly cover some aspects not previously discussed.

mboxes Mapping. MiddlePolice can leverage the end-user mapping [15] to achieve better mbox assignment, such as redirecting clients to the nearest mbox, mapping clients according to their ASes, and load balancing.

Incorporating Endhost Defense. MiddlePolice can cooperate with the DDoS defense mechanism deployed, if any, on the victim. For instance, via botnet identification [27,36], the victim can instruct the mboxes to block botnet traffic early at upstream so as to save more downstream bandwidth for clients. Such benefits are possible because the policies enforced by MiddlePolice are completely destination-driven.

Additional Monetary Cost. As discussed in 8.2.1, MiddlePolice introduces small computational overhead. Compared with basic DDoS-as-a-service solutions, MiddlePolice offers additional functionalities such as enabling destination-chosen policies and filtering bypassing traffic. In a competitive marketplace, service's price (the monetary cost) should scale with the cost of providing that service, which, in the case of MiddlePolice, is low.

11. CONCLUSION

This paper presents MiddlePolice, a DDoS defense system that is as deployable as cloud-based systems and has the same destination-based control of capability-based systems. In its design, MiddlePolice explicitly addresses three challenges. First, MiddlePolice designs a capability mechanism that requires only limited deployment from the cloud, rather than widespread Internet upgrades. Second, MiddlePolice is fully destination-driven, addressing the shortcomings of the existing capability-based systems that can work only with a single fairness regime. Finally, MiddlePolice addresses the traffic-bypass vulnerability of the existing cloud-based solutions. Extensive evaluations on the Internet, testbed and large scale simulations validate MiddlePolice's deployability and effectiveness in enforcing detestation-chosen policies.

12. ACKNOWLEDGMENTS

We thank the anonymous CCS reviewers for their valuable feedback. This material is based upon work partially supported by NSF under Contract Nos. CNS-0953600, CNS-1505790 and CNS-1518741. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, the University of Illinois, or the U.S. Government or any of its agencies.

13. REFERENCES

[1] PlanetLab. https://www.planet-lab.org/. WebCite archive.

[2] AT&T Denial of Service Protection. http://soc.att.com/1IIlUec, Accessed in 2016. WebCite archive.

[3] NS-3: a Discrete-Event Network Simulator. http://www.nsnam.org/, Accessed in 2016.

[4] AS Relationships – CIDR Report. http://www.caida.org/data/as-relationships/, Accessed in Dec 2015. WebCite archive.

[5] AS Names - CIDR Report. http://www.cidr-report.org/as2.0/autnums.html, Dec 2015.

[6] Akdemir, K., Dixon, M., Feghali, W., Fay, P., Gopal, V., Guilford, J., Ozturk, E., Wolrich, G., and Zohar, R. Breakthrough AES Performance with Intel® AES New Instructions. *Intel white paper* (2010).

[7] Andersen, D. G. Mayday: Distributed Filtering for Internet Services. In *USITS* (2003).

[8] Andersen, D. G., Balakrishnan, H., Feamster, N., Koponen, T., Moon, D., and Shenker, S. Accountable Internet Protocol (AIP). In *ACM SIGCOMM* (2008).

[9] Anderson, T., Roscoe, T., and Wetherall, D. Preventing Internet Denial-of-Service with Capabilities. *ACM SIGCOMM* (2004).

[10] Appenzeller, G., Keslassy, I., and McKeown, N. *Sizing Router Buffers*. ACM SIGCOMM, 2004.

[11] Argyraki, K., and Cheriton, D. Network Capabilities: The good, the Bad and the Ugly. *ACM HotNets-IV* (2005).

[12] Argyraki, K. J., and Cheriton, D. R. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *USENIX ATC* (2005).

[13] Basescu, C., Reischuk, R. M., Szalachowski, P., Perrig, A., Zhang, Y., Hsiao, H.-C., Kubota, A., and Urakawa, J. SIBRA: Scalable Internet Bandwidth Reservation Architecture. *NDSS* (2016).

[14] Briscoe, B. Tunnelling of Explicit Congestion Notification. RFC 6040, 2010.

[15] Chen, F., Sitaraman, R. K., and Torres, M. End-User Mapping: Next Generation Request Routing for Content Delivery. In *ACM SIGCOMM* (2015).

[16] Chiu, D.-M., and Jain, R. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN systems* (1989).

[17] Dimitropoulos, X., Krioukov, D., Fomenkov, M., Huffaker, B., Hyun, Y., Riley, G., et al. AS Relationships: Inference and Validation. *ACM SIGCOMM CCR* (2007).

[18] Dixon, C., Anderson, T. E., and Krishnamurthy, A. Phalanx: Withstanding Multimillion-Node Botnets. In *NSDI* (2008).

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

[20] Gao, L., Griffin, T. G., and Rexford, J. Inherently Safe Backup Routing with BGP. In *IEEE INFOCOM* (2001).

[21] Gilad, Y., and Herzberg, A. LOT: a Defense against IP Spoofing and Flooding Attacks. *ACM TISSEC* (2012).

[22] Gilad, Y., Herzberg, A., Sudkovitch, M., and Goberman, M. CDN-on-Demand: An Affordable DDoS Defense via Untrusted Clouds. *NDSS* (2016).

[23] Goldberg, S., Schapira, M., Hummon, P., and Rexford, J. How Secure are Secure Interdomain Routing Protocols. *ACM SIGCOMM CCR* (2011).

[24] Gueron, S. Intel Advanced Encryption Standard (AES) Instructions Set. *White Paper, Intel* (2010).

[25] Herbert, T. UDP Encapsulation in Linux. In *The Technical Conference on Linux Networking* (2015).

[26] Ioannidis, J., and Bellovin, S. M. Implementing Pushback: Router-Based Defense Against DDoS Attacks. *USENIX NSDI* (2002).

[27] Karasaridis, A., Rexroad, B., and Hoeflin, D. Wide-scale Botnet Detection and Characterization. In *USENIX HotBots* (2007).

[28] Keromytis, A. D., Misra, V., and Rubenstein, D. SOS: Secure Overlay Services. *ACM SIGCOMM* (2002).

[29] Kim, D., Chiang, J. T., Hu, Y.-C., Perrig, A., and Kumar, P. CRAFT: A New Secure Congestion Control Architecture. In *ACM CCS* (2010).

[30] Kührer, M., Hupperich, T., Rossow, C., and Holz, T. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *USENIX Security Symposium* (2014).

[31] Kuzmanovic, A., and Knightly, E. W. Low-Rate TCP-Targeted Denial of Service Attacks: the Shrew vs. the Mice and Elephants. In *ACM SIGCOMM* (2003).

[32] Liang, J., Jiang, J., Duan, H., Li, K., Wan, T., and Wu, J. When HTTPS Meets CDN: A Case of Authentication in Delegated Service. In *IEEE S&P* (2014).

[33] Liu, X., Li, A., Yang, X., and Wetherall, D. Passport: Secure and Adoptable Source Authentication. In *USENIX NSDI* (2008).

[34] Liu, X., Yang, X., and Lu, Y. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. In *ACM SIGCOMM* (2008).

[35] Liu, X., Yang, X., and Xia, Y. NetFence: Preventing Internet Denial of Service from Inside Out. *ACM SIGCOMM* (2011).

[36] Livadas, C., Walsh, R., Lapsley, D., and Strayer, W. T. Using Machine Learning Technliques to Identify Botnet Traffic. In *IEEE LCN* (2006).

[37] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. Controlling High Bandwidth Aggregates in the Network. *ACM SIGCOMM* (2002).

[38] Mittal, P., Kim, D., Hu, Y.-C., and Caesar, M. Mirage: Towards Deployable DDoS Defense for Web Applications. *arXiv preprint arXiv:1110.1060* (2011).

[39] Miu, T. T., Hui, A. K., Lee, W., Luo, D. X., Chung, A. K., and Wong, J. W. Universal DDoS Mitigation Bypass. *Black Hat USA* (2013).

[40] Naylor, D., et al. XIA: Architecting a More Trustworthy and Evolvable Internet. *ACM SIGCOMM* (2014).

[41] Networks, A. Worldwide Infrastructure Security Report, Volume IX. https://www.arbornetworks.com/images/documents/WISR2016_EN_Web.pdf, 2016.

[42] Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B., and Hu, Y.-C. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. *ACM SIGCOMM* (2007).

[43] Peter, S., Javed, U., Zhang, Q., Woos, D., Anderson, T., and Krishnamurthy, A. One Tunnel is (often) Enough. In *ACM SIGCOMM* (2014).

[44] Savage, S., Wetherall, D., Karlin, A., and Anderson, T. Practical Network Support for IP Traceback. *ACM SIGCOMM* (2000).

[45] Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G., and Tyson, M. FRESCO: Modular Composable Security Services for Software-Defined Detworks. In *NDSS* (2013).

[46] Song, D. X., and Perrig, A. Advanced and Authenticated Marking Schemes for IP Traceback. In *INFOCOM* (2001).

[47] Sundaresan, S., De Donato, W., Feamster, N., Teixeira, R., Crawford, S., and Pescapè, A. Broadband Internet Performance: a View from the Gateway. In *ACM SIGCOMM* (2011).

[48] Vissers, T., Van Goethem, T., Joosen, W., and Nikiforakis, N. Maneuvering Around Clouds: Bypassing Cloud-based Security Providers. In *ACM CCS* (2015).

[49] Walfish, M., Vutukuru, M., Balakrishnan, H., Karger, D., and Shenker, S. DDoS Defense by Offense. In *ACM SIGCOMM* (2006).

[50] Yaar, A., Perrig, A., and Song, D. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE S&P* (2004).

[51] Yang, X., Wetherall, D., and Anderson, T. A DoS-limiting Network Architecture. In *ACM SIGCOMM* (2005).

[52] Zhang, X., Hsiao, H.-C., Hasker, G., Chan, H., Perrig, A., and Andersen, D. G. SCION: Scalability, Control, and Isolation on Next-Generation Networks. In *S&P* (2011).