The Case for In-Network Replay Suppression

Taeho Lee*
ETH Zürich
kthlee@inf.ethz.ch

Christos Pappas*
ETH Zürich
pappasch@inf.ethz.ch

Adrian Perrig
ETH Zürich
aperrig@inf.ethz.ch

Virgil Gligor
Carnegie Mellon University
gligor@cmu.edu

Yih-Chun Hu
UIUC
yihchun@illinois.edu

ABSTRACT

We make a case for packet-replay suppression at the network layer, a concept that has been generally neglected. Our contribution is twofold. First, we demonstrate a new attack, the router-reflection attack, that can be launched using compromised routers. In this attack, a compromised router degrades the connectivity of a remote Internet region just by replaying packets. The attack is feasible even if all packets are attributed to their sources, i.e., source authentication is in place, and our evaluation shows that the threat is pervasive—candidate routers for compromise are in the order of hundreds or thousands.

Second, we design an in-network mechanism for replay suppression. We start by showing that designing such a mechanism poses unsolved challenges and simple adaptations of end-to-end solutions are not sufficient. Then, we devise, analyze, and implement a highly efficient protocol that suppresses replayed traffic at the network layer without global time synchronization. Our software-router prototype can saturate a 10 Gbps link using only two CPU cores for packet processing.

1. INTRODUCTION

End-to-end replay detection and suppression has been studied for over three decades and practical mechanisms have been deployed in many client-server [1–5], and host-to-host applications [6–9].

In contrast, in-network replay detection and suppression has been generally considered unnecessary. For example, the end-to-end argument in network design states that since an end application will detect and suppress replayed packets if deemed necessary, replay suppression is unnecessary at the network layer [10]. In this paper, we show that despite this seemingly persuasive argument, in-network replay detection and suppression is becoming an indispensable network functionality, and we provide a highly efficient mechanism that can be used on commodity routers.

We begin by the following two observations: 1) The common assumption that routers are trustworthy no longer holds, as attackers are becoming increasingly interested and successful in compromising network infrastructure. Poor security practices [11–13] enable attackers to obtain access to routers. More significantly, the adoption of emerging technologies such as software-defined networking (SDN) enables attackers to compromise the network directly [14, 15]. 2) In-network source and content authentication [16–19] is *insufficient* to thwart a broad range of replay attacks enabled by compromised routers. In the following, we describe three adverse consequences of packet replays by compromised routers.

First, source authentication—counter-intuitive as it may sound—can help an attacker to *frame* an innocent source. For example, a compromised router can deliberately replay packets to cause abnormally high packet rates and trigger intrusion detection systems. Here, the adversary takes advantage of typical intrusion classification rules to falsely accuse a source of misbehavior; e.g., to make it appear malicious. Such attacks are particularly insidious, since the source has no readily available recourse; e.g., because traffic repudiation mechanisms require global inter-ISP cooperation [20], which is difficult to orchestrate across different jurisdictions.

Second, replaying packets can be used to deliberately waste network resources and corrupt accounting mechanisms. For instance, a system that allocates network resources (e.g., bandwidth) to authenticated sources [21, 22] can be easily overwhelmed by replaying authentic packets. Furthermore, to increase billable traffic on one of its underutilized paths, a malicious network (e.g., Tier-1 ISP) could compromise a router in an upstream network, replay authenticated traffic there, and then charge its customers for the artificially generated extra traffic.

Third, we show that the effects of potential attacks are not local, i.e., they do not affect only the implicated source(s). We present a new attack—the *router-reflection* attack—that enables an adversary to attack a geographic region of the Internet. The adversary uses a compromised router and leverages services that do not perform end-to-end replay detection (e.g., DNS or NTP): the attacker finds the *routing bottlenecks* of the target region [23] and replays requests whose responses will target these bottleneck links on the return path. The attacker can easily find such bottleneck links as they are both pervasive and hard to remove in the current Internet; and that these links are sufficiently provisioned only for a normal mode of operation, but not for targeted flooding [24, 25]. We dedicate Section 2 to the design and analysis of the attack.

In our quest to devise a practical in-network replay-suppression mechanism, we found that simple adaptations of well known end-to-end mechanisms cannot be used at the network layer: processing, storage and communication overheads, and time synchronization requirements raise numerous challenges.

Our in-network replay detection and suppression design is based on a combination of *per-interval sequence numbers* with small *rotating Bloom filters* that store observed packets for the currently

*These authors contributed equally to this work.

ASIA CCS '17, April 02-06, 2017, Abu Dhabi, United Arab Emirates

DOI: http://dx.doi.org/10.1145/3052973.3052988

active sequence-number window. Our design requires only minimal coordination between domains (the sequence-number-window update interval) and does not rely on global time synchronization. Furthermore, we optimize the protocol parameters to ensure very low overhead with respect to processing, storage, and communication latency. In fact, our software prototype demonstrates that in-network replay suppression is practical to perform even on commodity routers.

In-network replay suppression comes with further interesting repercussions: it ensures that every bit in transit is attributable to its actual source, which is necessary for all accounting mechanisms. Moreover, loops are inherently prevented, thus the Time-To-Live field is no longer necessary.

In summary, this paper makes the following contributions:

- illustrate attack capabilities enabled by in-network replays and evaluate their use in a new link-flooding attack (Section 2);
- show that traditional end-to-end replay detection is not suitable to prevent in-network replays (Section 3);
- define a new protocol for in-network replay detection and present its salient features (Section 4);
- evaluate the new protocol and show that it provides efficient and scalable replay detection (Section 5).

2. ROUTER-REFLECTION ATTACK

In this section, we describe the *router-reflection attack*, a new attack in which an adversary degrades, or blocks, legitimate traffic from flowing into a chosen geographic region of the Internet. The adversary compromises routers and replays packets in order to flood targeted links that carry a majority of routes into the region. The attack has similar goals as that of the Crossfire attack [24], but the strategy and the adversary's capabilities are different: it does not rely on large botnets; it focuses on responses from public servers, rather than requests to public servers.

2.1 Overview

Consider a set of hosts V, which are distributed over the Internet, and a set of hosts T inside a confined region of the Internet—the *target area*—against which the adversary launches the attack. A target area can include the hosts of a city, an organization, or even a small country. We refer to the traffic direction from V to T as the *inbound* direction and to its reverse as the *outbound* direction. The set of layer-3 links that carry a majority of routes from V to T are the *routing bottlenecks* of the target area. A routing bottleneck is different from a bandwidth bottleneck [26] in that a bandwidth bottleneck is determined by the traffic load, whereas a routing bottleneck is determined by the number of flows (source-destination pairs) that it carries. Typically, routing bottlenecks are adequately provisioned and the traffic flows do not experience degraded performance in the absence of flooding attacks. Henceforth, the term bottleneck refers to routing bottlenecks.

The goal of the adversary is to turn the routing bottlenecks of the target area into bandwidth bottlenecks and degrade the performance of as many flows as possible. To this end, the adversary compromises a router near the target area and replays observed traffic. Specifically, the adversary replays legitimate outbound requests from hosts in T to selected services of hosts in V that do not perform end-to-end replay detection (e.g., most UDP-based services). The corresponding responses from hosts in V hit the routing bottlenecks of the target area in the inbound direction and consume the bandwidth of these links (e.g., router R1 in Figure 1).

In its simplified version, the attack does not rely on traffic responses: a router can replay inbound traffic and hit routing bottlenecks that are located downstream (e.g., router R2 in Figure 1). Hence, a router can replay a larger portion of the observed flows – not only UDP-based services.

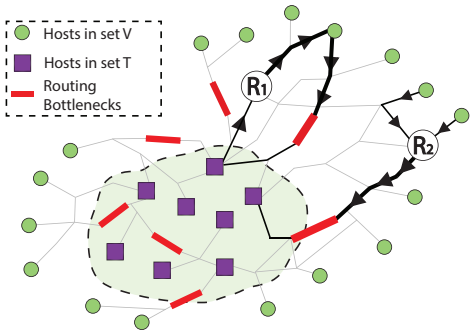

Figure 1: Router-Reflection Attack: compromised routers R1 and R2 can target routing bottlenecks by replaying legitimate traffic.

Our attack builds on intuition gained by recent work [23]: routing bottlenecks are target-area-specific, pervasive, and long-lived. Furthermore, the attack has three distinguishing characteristics.

1. It exploits the fact that services that do not perform replay detection are ubiquitous. There is an abundance of UDP-based services used for common tasks (e.g., DNS, SSDP, NTP) that will generate responses for replayed requests.
2. It does not inject "new" traffic nor does it modify the observed traffic. Thus, the attack does not require large botnets to create traffic and it is feasible even with source-authentication systems [16–18] in place. Note the difference from common reflection attacks that spoof the source address, directing the response traffic to a victim.
3. It exploits the fact that Internet paths tend to be *asymmetric*, especially when they traverse core backbone links [27,28]. This means that the responses generated by the replayed requests will likely follow a different inbound path back to the target area. Thus, a compromised router can launch such an attack without attacking itself in the inbound direction.

We emphasize that we assume a source-authentication scheme is in place. That is, a router can verify the authenticity of a packet (e.g., at the AS level) and drop modified and injected traffic. In the strict sense, source authentication should detect replayed packets as well, since the actual source of a replayed packet is the entity that injects the replayed packet. However, none of the source-authentication schemes handle in-network replay detection explicitly; this raises the new class of attacks that we describe in this section.

2.2 Execution

To launch a router-reflection attack against a target area, the adversary proceeds in four stages: first, she selects the set of hosts T and V; then, she computes the routing bottlenecks for the target area; next, she identifies candidate routers for compromise; and finally she uses a compromised router to replay packets of specific flows.

2.2.1 Stage 1: Selection of Host-Sets T and V

The adversary begins by selecting a set of public servers in a target area (set T). Furthermore, she selects a set of nodes that are geographically distributed across the globe and will act as vantage points for the target area (set V). Note that the hosts in V do not participate in the attack and are not under the adversary's control; they are used only to map the target area. The set V can be constructed using Looking Glass (LG) servers that are globally distributed. An

LG server is an Internet node that is accessed remotely (usually through a web interface) and runs a limited number of commands (e.g., traceroute and ping). For instance, CAIDA provides a list with approximately 1500 LG servers located in 77 different countries and 268 different ASes [29].

2.2.2 Stage 2: Routing-Bottlenecks Computation

In order to compute the bottleneck links, the adversary constructs a link-map that is centered at the target area and then computes the flow density for every link in the map. We briefly present this procedure, as it has been proposed in previous work [24].

Link-Map. To construct the link-map, the adversary performs traceroutes from all vantage points (set V) to all public servers in the target area (set T), which yields $|V| \cdot |T|$ distinct traces. A trace consists of a sequence of IP addresses that belong to the interfaces of the routers on the path. The IP addresses of two adjacent routers' interfaces define a link. Thus, using all obtained traces, we get a link-map centered at the target area.

The computed link-map includes unstable routes that must be eliminated. In order to increase reliability and resource utilization, routers are often configured to load-balance their traffic over multiple paths; e.g., using per-flow or even per-packet policies [30, 31]. Thus, for the same source-destination pair, some links appear always in the traces— *persistent links*—and some do not—*transient links*.[1] The adversary eliminates transient links from the link-map, as they do not qualify for candidate routing bottlenecks: it is unclear whether and under which conditions replayed traffic can indeed reach a transient link.

Flow-Density. Given the link-map and the traces, the adversary computes the flow density for each persistent link, i.e., the number of flows that traverse the link. A high flow density for a link means that it carries a large number of the generated traces and is an indicative metric of the overall number of flows as well.

Routing bottlenecks are determined by sorting the links in a descending order of flow density and then selecting the b highest ranked links. Higher values of b mean that more links (and thus more flows) can be considered. However, attacking only a few links is sufficient to affect a large fraction of the inbound traffic and achieve the adversary's goal.

2.2.3 Stage 3: Attack-Router Selection

In the third stage of the attack, the adversary discovers candidate routers for compromise. The adversary will then try to compromise routers that can target as many bottlenecks as possible.

Routers for outbound replay. The adversary discovers routers that can replay outbound traffic whose inbound responses will traverse one or more bottlenecks.

To execute this step, the adversary performs traceroutes from nodes in the target area to all hosts in V. The goal of the step is to discover as many interfaces (and thus candidate routers) as possible; thus, interfaces that perform load balancing are not eliminated. Furthermore, the adversary must perform alias resolution for the discovered interfaces, since the goal is to identify routers—not links as in the inbound direction. Note that the adversary does not control nodes in the target area, but there is a number of options to perform this step. For example, she can use an LG server that is located in the target area; or she can issue reverse traceroutes [32] to hosts in V; or use existing tools to discover the topology of an ISP [33].

[1]The traceroute dataset for our experiments (See Section 2.3) contains 2.3 million links, 44.6% of which are persistent.

Routers for inbound replay. For the simpler version of the attack, the adversary uses the traceroutes from Stage 1. Using the traces from V to T, the adversary locates the interfaces (and with alias resolution the corresponding routers) that can replay packets and target bottlenecks downstream.

Our evaluation (Section 2.3) follows the first three stages of the attack and demonstrates that candidate routers are in the order of hundreds or thousands.

2.2.4 Stage 4: Packet Replay

In the final stage, the adversary has compromised one or more of the candidate routers and launches the attack. The adversary follows a similar procedure as in Stage 3, but this time using the actual observed traffic. For outbound traffic, the adversary determines which flows will result in responses that will traverse bottleneck links and ensures that she is not on the inbound path. To gain insight about the reverse path, she can use similar methods as described in Stage 3 (e.g., LG servers and reverse traceroute). For inbound traffic, the adversary must determine which of the flows can be replayed in order to target a bottleneck link that is located downstream. The adversary can simply traceroute to the destination of the flows and compare the traces with the bottlenecks computed in Stage 1. In Section 2.4, we discuss more practical considerations for launching the attack in both directions.

2.3 Experimental Results

In this section, we show that the router-reflection attack is practical; that is, we show that for a chosen target area there is an abundance of candidate routers that can be compromised to attack routing bottlenecks. Our chosen target areas {*Area1, Area2, Area3, Area4*} are a permutation of the alphabetically ordered list {*Japan, Rome, Seoul, Singapore*}. We emphasize that the feasibility and severity of the attack is not target-area specific since routing bottlenecks are an elemental property of today's Internet due to route-cost minimization [23]; thus, our findings are not limited to the above-mentioned areas.

In our experimental setup, we follow Stages 1-3 as described in Section 2.2. For Stages 1-2, we use approximately 200 Planetlab nodes as our vantage points, which are distributed in 34 different countries and 97 different ASes. We choose 1000 public servers in the target area using a public search engine with geolocation properties.[2] Furthermore, we vary the number b of links that we consider as bottleneck links from 20 to 40. For Stage 3, we choose a small number of measurement points in the target area that will be used to perform traceroutes to the vantage points; we obtain the measurement points from RIPE Atlas [34]. Finally, we perform traceroutes from all Planetlab nodes to all nodes in the target area, and from all measurement points to all Planetlab nodes. This gives us both the list of routing bottlenecks and the list of candidate routers.

Routing Bottlenecks. The first interesting result is the location of the routing bottlenecks in terms of hop distance from the vantage points and the target area. We measure the average hop distance from the source and compare it to the average path length. For many of our traces, we do not obtain responses from the last hops; usually this is due to firewalls in the hosts' local networks. In such cases, we assume that the destination resides after the last responding hop, resulting in a shorter average path length. In other words,

[2]We used SHODAN (https://www.shodan.io/) as our search engine. When target areas are more confined regions (e.g., a city), the location of the public servers must be cross-verified with other geolocation services.

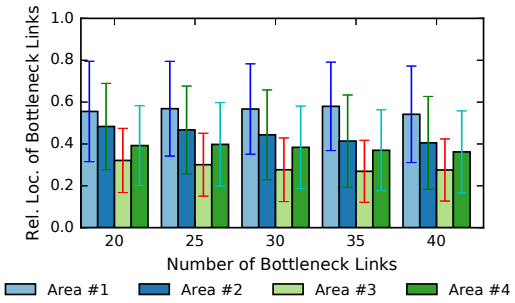

Figure 2: Relative location and standard deviation of bottleneck links.

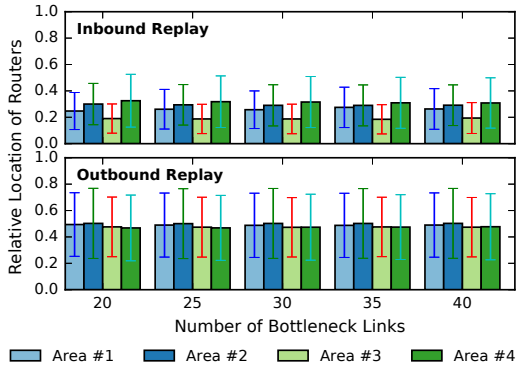

Figure 3: Relative location and standard deviation of routers that can target at least one bottleneck, for inbound and outbound replays.

we obtain an upper bound for the relative location of the bottlenecks with respect to the average path length.

Figure 2 shows the average relative location and standard deviation of the bottleneck links for each target area. The result shows that the routing bottlenecks are located approximately in the middle of the routes and confirms the results of previous work [23]. Furthermore, the location of the links does not fluctuate significantly as the number of bottleneck links increases.

Attack Router Identification. We discover routers that can replay packets in the outbound and inbound direction, and hence, are candidates for compromise. We show the average location of candidate routers and the number of routers that can target at least one bottleneck link.

Figure 3 shows the average location of the candidate routers that can replay packets; the upper and lower box plots show the location of the routers for inbound and outbound replay, respectively. For inbound replay attacks, the candidate routers are located before the bottleneck links; this is expected, since bottleneck links must be located downstream with respect to the candidate routers. For outbound replay attacks, the candidate routers are located approximately in the middle of the routes; this happens because route diversity increases close to the core, and thus, routers can launch attacks without attacking themselves in the inbound direction. Again, we see no considerable change as b changes.

Figure 4 shows the number of candidate routers that can target at least one routing bottleneck; the upper and lower portions of each bar represent the number of candidate routers that can be used for outbound and inbound replay attacks, respectively. We observe two interesting findings. First, there is an abundant number of candidate routers to compromise, ranging from hundreds to thousands. Second, increasing the value of b does not significantly increase the

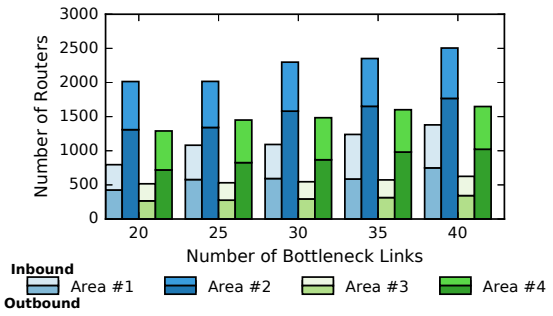

Figure 4: Number of routers that can target at least one bottleneck.

number of candidate routers. This is because the additional links that are considered are adjacent to the initial routing bottlenecks, and thus, only a few more candidate routers are discovered.

2.4 Practical Considerations

Mitigating Measurement Inaccuracies. We had to handle two common sources of inaccuracies related to traceroutes.

First, traceroute may miss nodes, links, or even report false links [35]. We do not use specialized traceroute tools (e.g., Paris traceroute [35]), since load-balanced links cannot become routing bottlenecks. Instead, we obtain multiple traces for every flow (10 probe packets per trace) to eliminate inaccuracies due to load balancers.

Second, alias-resolution tools are mostly based on implementation specific details of routers and may introduce false negatives and false positives. False negatives fail to cluster interfaces that belong to the same router, which may reduce the number of candidates for attack routers. However, this is not an issue since we can still find plenty of candidate routers. False positives associate interfaces of different routers to the same router, which can lead to false router identification as good targets. To reduce/eliminate false positives, we use the Monotonic ID-Based Alias Resolution (MIDAR) [36] tool from CAIDA for two reasons. First, the monotonic bound tests of MIDAR yield a very low false positive rate. Second, its efficiency in resolving aliases in large lists of interface IP addresses: to resolve aliases in a list of N IP addresses it probes $O(N)$ pairs instead of testing the $O(N^2)$ candidate pairs.

Compromising Routers. In this paper, we focus on one severe consequence of router compromises rather than software security of routers; the latter is a research topic on its own right.

Our work, however, is motivated by the observation that compromised routers are already a major concern for ISPs. It is known that state-level adversaries are massively targeting routers—they are easy to compromise as they are rarely updated and lack security software to detect breaches [37]. Cisco has issued a document to warn operators of attacks against their routers and to inform them about commonly used attack vectors [38]. Security companies consider these attacks only the tip of the iceberg and highlight the difficulty of detecting such compromises, allowing attackers to maintain access for long time periods [39].

In addition, the emergence of SDN in ISP networks provides endless possibilities for infrastructure compromises, since SDN security is not yet mature. Researchers have warned that attackers can compromise networks directly [14,15]: First, controller vulnerabilities allow attackers to compromise controllers and take control over the *entire* underlying infrastructure [40]. Second, SDN switches have proven insecure as well, allowing attackers to install persistent malware [41].

Amplification Effect. Our attack leverages UDP services since they do not perform replay suppression and they commonly have responses that are much larger than the requests that caused them, i.e., we exploit the UDP amplification effect. Recent attacks have exploited extreme amplification factors (e.g., NTP monlist commands can have a factor up to 4700) of misconfigured services, however, moderate amplification factors are common in legitimate requests as well. For example, DNSSEC requests can have an amplification factor of 30; the GetBulk operation in SNMPv2 has an amplification factor of 6.3; and the BitTorrent hash searches have an average factor of 3.8. Routers that replay packets in the inbound direction have to rely on their available capacity to cause congestion. Note that although bottleneck links are adequately provisioned for normal network conditions, the additional load caused by small-to-moderate amplification at a router's full capacity would significantly degrade the available capacity.

Early Congestion. In case of early congestion, a link that is located upstream of the bottleneck link gets congested. Early congestion does not render the attack impossible, but confines its effect. A router that replays outbound traffic has no visibility in the inbound direction and thus cannot react to early congestion. A router that replays inbound traffic, however, has the bottleneck links located downstream. Thus, the router can perform traceroutes to the target area and determine early congestion based on the responses: in case of early congestion the router would not receive most of the ICMP replies. The router can then react by decreasing the replay rate of the corresponding flows; at the same time, it can increase the replay rate of flows that exit from the same interface, but hit another bottleneck.

Attack Detectability. A high replay rate of specific flows can trigger firewall alarms whenever network operators employ rate-limiting controls; e.g., Response Rate Limiting in DNS name servers. Although, in principle, this may limit an adversary's high-intensity replays for outbound, in practice this is easily overcome: an attack router can replay different flows that hit the same routing bottlenecks in the inbound direction.

Replaying packets in the inbound direction (see simplified attack) is less prone to rate-limiting. Intrusion detection systems and protection mechanisms are typically deployed close to the hosts and are not pervasive in the network, offering protection to resources of end systems, but not network resources. Routing bottlenecks are located in the middle of the routes and thus at a safe distance from the actual targets of the attack. This yields such defense mechanisms ineffective against inbound packet replay.

3. CHALLENGES FOR IN-NETWORK REPLAY SUPPRESSION

End-to-end replay-suppression mechanisms, i.e., mechanisms realized at the communicating end hosts, cannot be used at the network layer due to fundamental operational differences:

1. The packet throughput of routers is orders of magnitude higher than the packet throughput of the fastest services. Consequently, a replay-detection overhead that is tolerable for a server may be intolerable for a router.
2. Routers are equipped with a few tens of MBs of fast memory (SRAM) per data-path chip. Routers use fast memory for per-packet operations in order to minimize latency and sustain a high throughput. However, hardware manufacturers do not integrate more than a few tens of MBs of memory per chip, as the yield becomes too low to sustain the manufacturing process. On the contrary, end servers can meet their performance requirements by using larger and slower memory (DRAM) combined with the fast memory of CPU caches.
3. Novel mechanisms at the network layer often require coordination and introduce complex interactions between network entities. For example, a mechanism may require time synchronization among routers of different domains.

3.1 In-Network Mechanisms

To detect and suppress replayed packets in the network, a router must inspect each packet it forwards; thus, the detection mechanism must be efficient and lightweight so that it does not impair forwarding performance. We consider three main challenges for universally deploying a novel mechanism at the network layer:

1. **Computation Overhead.** In order to sustain a high throughput, a router has a strict time budget to serve a packet. The two major components that carve out this budget are latencies due to memory operations and due to CPU-intensive operations. For example, if the memory footprint for replay detection is too large to fit into the fast memory of the router (e.g., on-chip caches), forwarding performance will be degraded due to cache misses. Likewise, if CPU-intensive operations, such as frequent public-key signature verifications become necessary, the forwarding performance will suffer.
2. **Communication Overhead.** The communication overhead comes in the forms of latency overhead and bandwidth overhead. For example, if a router needs to ask another entity (e.g., the source host who created the packet or a remote router) to check the authenticity of the packet, the communication latency will increase substantially. Furthermore, additional data in packets or additional messages will increase the bandwidth overhead (especially if they are sent frequently).
3. **Time Synchronization.** In one extreme, a replay detection mechanism does not require any of the entities (e.g., routers, ASes, hosts) to be synchronized; and on the other extreme, it may require every entity in the Internet to be synchronized. A middle-ground solution may require only parts of the networks to be synchronized (e.g., entities within each autonomous system).

3.2 Inadequacy of E2E Replay Detection

Early work on replay detection identified four basic primitives, and combinations thereof, that are found in all end-to-end protocols for secure communication [42]. We present these primitives and study their applicability for in-network replay detection.

Storing Packet Digests. A router stores a digest for every packet that it forwards. When the router receives a new packet, it checks whether it has seen the packet before. This has significant advantages: it has a relatively low processing overhead (i.e., a single hash operation), no communication overhead, and no time-synchronization requirement. However, it has a large memory footprint, which makes it impractical for routers: for a fully saturated 10-Gbps link, a router needs to store 10^9 bits of data for each passing second. Even an efficient storage data structure, such as a Bloom filter is impractical: a router would need 142 MB to store packets received in one minute, assuming a target false positive rate of 10^{-5} and the largest packet size of 1500-byte, i.e., the lowest packet rate. No router can store 142 MB in its on-chip cache. Even if the router stores packets for a minute, adversaries can still replay packets after one minute. Thus, indefinitely storing observed traffic without a mechanism to discard packets is not viable.

Sliding Time Windows. A router maintains a time window and accepts packets whose timestamps fall within the window. This requires time synchronization since the source needs to use a times-

tamp that falls within the time window of the router. To prevent replays, the router needs to store the packets that it has forwarded in a buffer until the packets become invalid by falling out of the sliding time window. This approach has minimal communication overhead (the timestamp in the packet) since the router does not exchange additional messages with the source. However, minimizing the size of the buffer comes at the cost of strict time synchronization so that legitimate packets are not dropped. Since precise Internet-wide time synchronization is impractical, we consider this approach also impractical.

Per-Packet Sequence Numbers. A source and an intermediate router maintain a sequence number for the source's last observed packet: the source inserts a sequence number in each packet, and the router accepts packets that have a higher sequence number than the router has previously seen from the source. This approach does not require any time synchronization, does not incur any latency overhead, and does not introduce prohibitive computation overhead. However, packet reordering can cause dropping of legitimate packets, when packets with higher sequence numbers arrive before packets with lower sequence numbers. Furthermore, this mechanism requires per-source state at routers; thus, the storage overhead depends on the granularity at which sources are identified. For example, if sources are identified at the granularity of a host, then this approach requires per-host state at routers, which is impractical.

Challenge-Response. For each packet, a router asks the source host to send a proof of transmission that verifies that the packet is not a replay: the router inserts a nonce and expects from the source a cryptographic signature over the nonce; alternately, the source can produce a message authentication code (MAC) using a key that is shared with the router. The latter approach has a relatively small computation overhead and does not require time synchronization. However, it has the largest communication overhead of all mechanisms since a separate challenge is needed for every packet that traverses every router. Hence, it is impractical.

Second Chance. This hybrid approach [42] combines three of the four primitives discussed above: it uses a variable-sized sliding time window and uses a buffer to store past packets for the duration of the window. To eliminate the requirement for time synchronization between a source and a router, it uses a challenge-response mechanism when needed: when the packet's timestamp falls outside the router's time window, the router asks the source to resend the packet using a timestamp the router provides, thereby giving the source a second chance. This approach is impractical at the router level because packets typically go through multiple routers that perform replay detection, and as a result, a packet may experience multiple rounds of second chances; thus incurring a large communication overhead that makes the approach impractical.

4. IN-NETWORK REPLAY SUPPRESSION

We present our solution for in-network replay suppression, starting with our assumption. Then, we provide a high-level overview of the solution, followed by the protocol details. Finally, we formulate an optimization problem to determine all the parameters.

Assumption: We assume that a source-authentication scheme is deployed, i.e., every packet in the network is attributed to its source. We emphasize that replay attacks are meaningful *only* if source authentication is deployed; otherwise, an adversary controlling a router can directly inject/spoof traffic and attack any target. Therefore, source authentication is a fundamental requirement of such security schemes; it is not a specific limitation of our mechanism. As we show in Section 2, source authentication does not prevent replay attacks, however it is necessary to prevent malicious routers from tampering with traffic.

The source-authentication literature has several proposals that can be used. For instance, OPT [18] uses dynamically re-creatable keys in order to provide scalable source and data authentication. Packet Passport [17,43] uses multiple message authentication codes (MACs) to provide AS-AS authentication; each MAC is computed with the shared-key between the source AS and the transit AS on the path. Shared keys are generated through Diffie-Hellman key exchanges, using the public and private keys of ASes; the public keys are obtained from RPKI [44]. OPT and Passport are practical at the router level and can be used by our mechanism.

4.1 Overview

We build our replay-suppression mechanism based on two primitives: *per-interval sequence numbers* and *storing packet digests* in a Bloom Filter (BF). A valid sequence number is the first control check to ensure that a packet is legitimate (Section 4.1.1). If the packet is accepted, it is then checked against a locally stored list of previously observed packets that also have valid sequence numbers (Section 4.1.2). In other words: packets that are stored and replayed significantly after their observation time will be caught due to a sequence-number mismatch; packets that are replayed shortly after their observation will be caught by the BF; and packets that are replayed with a modified sequence number will be caught by source authentication. Through this combination, we build a mechanism that does not require global time synchronization and does not introduce communication overhead due to additional messages. Furthermore, our mechanism does not require adoption at every router, but can be deployed only at border routers of ASes; we discuss the security implications of the deployment locations in Section 6.1.

4.1.1 Per-Interval Sequence Numbers

Recall from Section 3 that the use of per-packet sequence numbers (seqNos) has two implications for a replay-suppression mechanism: the storage overhead depends on the granularity at which sources are identified, and that legitimate packets may be dropped if a packet with a higher seqNo arrives earlier than a packet with a lower seqNo.

In our approach, every source AS uses a seqNo, and every other AS (more precisely, their border router) remembers the seqNo of the source AS; in other words, routers keep per-AS state. More precisely, the source AS embeds a seqNo in every outbound packet, and transit routers only accept packets with seqNos that fall within a seqNo window. Furthermore, the source AS does not increment its seqNo per packet, but at fixed time intervals; in essence, ASes achieve loose synchronization without relying on global time synchronization. Our approach raises two important issues: the update frequency of the seqNos and the dissemination mechanism for new seqNos.

Update Frequency. The source AS periodically increments its seqNo in order to invalidate previously sent packets with smaller seqNos. Note that to achieve loose synchronization, the seqNo-update frequency is the only parameter that requires global agreement among ASes: ASes update their seqNos at a constant interval, but the seqNo values and the actual events of updating them are not synchronized. This approach makes it easier to handle packet reordering. A router maintains a seqNo window and only the packets with seqNos within the window are accepted. The use of per-packet seqNos makes it hard to determine an appropriate length for the window so that legitimate packets are not dropped; the length depends on parameters that change dynamically over time, such as traffic patterns (e.g., packet bursts) and load balancing at interme-

Table 1: Summary of Parameters and Notation

Parameters determined by the environment	
r	Incoming packet rate at the routers.
σ	Maximum latency variation between packets.
Parameters determined by the optimization problem (§4.3)	
T	Sequence-number-update interval.
fp	False-positive rate.
M	Length of the sequence-number window.
L	Bloom filter switching interval.
Δ	Additional time to delay sequence-number updates.
N	Number of bloom filters.
m	Size of bloom filter.
k	Number of hash functions or bloom filter indices.
Symbols for Source AS S	
SN_S	Sequence number used by S.
Symbols for Router R	
SN_S^R	Sequence number that router R maintains for S
TTL_S	TTL until R increments SN_S^R.
BF_i	i-th bloom filter, where $0 \leq i < N$.
BF_w	Bloom filter where incoming packets are inserted.
Other Symbols	
p	An arbitrary packet.
SN_S^p	Sequence number of S that is encoded in packet p.

diate routers. With our approach we mask away this complexity, since we only need to consider the maximum variance in one-way latency (more details in Section 4.3).

Update Dissemination. Transit routers must be informed and keep up-to-date information for the valid seqNos of every AS, so that the observed packets can be checked. To this end, we use the following two mechanisms.

The first one is an in-band mechanism: the source AS increments its seqNo, which is carried in outbound packets, and thereby informs routers in other ASes. Upon receiving a packet, router R updates the locally stored seqNo SN_S^R for the source AS S, if the packet is authentic and it carries a higher seqNo SN_S^p than the one stored. The benefit of this approach is that it does not require any additional messages to disseminate updated seqNos.

The second one is a local mechanism: a router increments the locally stored seqNo for the source AS by itself, if a new seqNo is not seen for a prolonged period of time; we refer to the time interval after which the seqNo for the source is incremented as TTL_S. This self-initiated update is necessary to limit the storage overhead at routers, since a router stores all packets with valid seqNos to prevent replays (Section 4.1.2); if the seqNo for a source is not updated, the router cannot delete the stored packets from the source. Thus, TTL_S will be a function of the seqNo-update interval, so that the router's seqNo for the source closely follows the source's seqNo in every case.

4.1.2 Storing Packet Digests

A router stores digests for previously observed packets to guarantee that packets with valid seqNos are not replayed. This is a consequence of using per-interval seqNos, since a seqNo does not uniquely identify a packet.

We create a data structure that consists of multiple BFs that are periodically rotated. In this data structure, a packet is inserted only into one of the filters, which we denote as the *writeable* filter; however, when searching if the packet has been previously observed, all filters are searched. Furthermore, the filters are periodically rotated in a round-robin fashion to prevent flooding of a single filter with too many insertions.

Algorithm 1 Processing of Packets p at Router R

$authenticate(p)$	Checks if p is authentic
$get\_src\_info(p)$	Retrieves SN_S^R, TTL_S for S
$bf\_lookup(p, BF)$	Lookup p in bloom filter BF
$bf\_insert(p, BF)$	Insert p into bloom filter BF

```
1: if !authenticate(p) then
2:    return
3: end if
4: {SN_S^R, TTL_S} ← get_src_info(p)
5: if SN_S^p < SN_S^R − M then
6:    return
7: else
8:    if SN_S^p > SN_S^R then
9:       SN_S^R ← SN_S^p
10:      TTL_S ← T + Δ
11:   else
12:      for 0 ≤ i < N − 1 do
13:         if bf_lookup(p, BF_i) then
14:            return
15:         end if
16:      end for
17:   end if
18:   bf_insert(p, BF_w)
19:   Forward p
20: end if
```

We emphasize that a packet is inserted to a BF independent of the seqNo in the packet; the packet is added only to the currently active filter, the writeable filter. However, the observed packet is checked for replay by checking all BFs for membership, including the writeable one; a positive response from any of the filters indicates a replay. In order to delete packets, the filter that becomes writeable is reinitialized to zero. Note that packets in the zeroed filter have sequence numbers that are no longer valid and will be discarded if replayed. This approach naturally raises two inter-related issues: how to determine the number N of filters in the data structure and the frequency of rotation L.

Recall that a router periodically rotates the BFs, but the rotation is independent of the seqNo updates by the ASes. In order to ensure replay detection, a router must remember a packet at least until the packet seqNo is invalidated. That is, the BF coverage period must be at least as long as that of the seqNo-window, so that valid seqNos cannot be replayed. Hence, the time window that the BFs must cover, which is $N \cdot L$, must exceed the amount of time that is needed for the packet to become invalid. At the same time, BFs must be small to reduce storage requirements and fit in fast memory. In Section 4.3, we take N and L into account to compute the optimal parameters for our replay-suppression protocol.

4.2 Protocol Operations

We present the tasks that are performed by the egress border routers of the source AS S and by the ingress border routers R of the intermediate and destination ASes. Table 1 summarizes the parameters and the notation we use.

Source AS: S inserts its seqNo SN_S in every outbound packet. In addition, it increments its seqNo SN_S after each interval T.

Ingress Router: For each incoming packet from a neighboring AS, R checks if the packet falls within the seqNo window for S and if the packet is present in any of the BFs. Algorithm 1 describes the procedure that R executes for incoming packets.

Initially, the router checks the authenticity of the packet (Lines 1-2) and checks if the seqNo in the packet (SN_S^p) falls within the seqNo window (Lines 5-6). If the packet is not authentic or has an invalid seqNo, the packet is dropped and the procedure terminates. Then, for valid packets, the router checks if the packet has been previously recorded in any of the N BFs (Lines 12-16). If the packet has not been seen previously, it is added to the writeable filter (Line 18) and then it is forwarded (Line 19). Note that when querying/adding a packet to a BF, the router computes a pseudo-random-function (PRF) over the content of the packet, excluding the mutable packet fields. The output of the PRF is used to determine the bits in the BFs that must be checked/set; the key for the PRF is known only to the AS. The use of a PRF is necessary to prevent an adversary from launching a chosen insertion attack against the BFs: if the adversary can control which bits in the writeable filter are set, it can set all bits in the BFs and cause *all* packets to be recognized as replays.

Furthermore, if the seqNo in the packet is higher than the one locally stored for S, R updates its seqNo and reinitializes TTL_S to $T+\Delta$ (Lines 8-10), which is the count-down timer used to self-update the seqNo of the source. Δ is an additional short delay that is used to ensure that the router does not increment the source's seqNo faster than the source does. If a router were to increment the source's seqNo faster, then it could be that after a long time period packets from the source AS would be dropped.

In addition, R performs one more task (not described in Algorithm 1). It periodically decrements TTL_S for each AS; if TTL_S becomes zero, the seqNo of the corresponding AS (SN_S^R) is incremented and TTL_S is reset to $T+\Delta$. This implements the count-down timer that is used for the self-updates of the sources' seqNos.

4.3 Optimization Problem

In this section, we formulate an optimization problem that involves the inter-related parameters of our mechanism. An appropriate configuration of the parameters and especially of the BFs is crucial to guarantee a high forwarding performance for the routers. We describe performance as a function of all the involved parameters: $f(m,k,N,L,M)$. Then, we derive constraints between the parameters, which gives us the following optimization problem:

$$\begin{aligned}
&\text{minimize} && f(m,k,N,L,M) \\
&\text{subject to} && M > \frac{\sigma}{T}+1, && (1)\\
& && N > \frac{1.1(T+\sigma)}{L}+1, && (2)\\
& && m > \frac{-krL}{\ln\left(1-(1-(1-fp)^{\frac{1}{N}})^{\frac{1}{k}}\right)}, && (3)\\
& && m,k,N,L,M \in Z^+ && (4)
\end{aligned}$$

Sequence-number update interval (T). T represents the time period for which a seqNo is used, before it is incremented. We consider values on the order of a few milliseconds (e.g., 10ms), and we show that it leads to an efficient implementation.

Additional delay window (Δ). Δ is a delay period that is used to slow down the update rate of router's R view for the seqNo of AS S (SN_S^R). It ensures that R does not increment its seqNo faster than S, i.e., does not increment at an interval shorter than T.

The main reason for an early update is the clock drift between S and R. We define Δ with respect to T, since the amount of clock drift is proportional to the time period under consideration; we are interested in estimating the seqNo-update inaccuracy for SN_S and SN_S^R by S and R respectively. We conservatively assume that the clock drift is lower than $0.05 \cdot T$; Marouani et al., report that a clock variation of 0.5 ppm, i.e., a drift of 0.5 µs per second, is a conservative estimate [45].

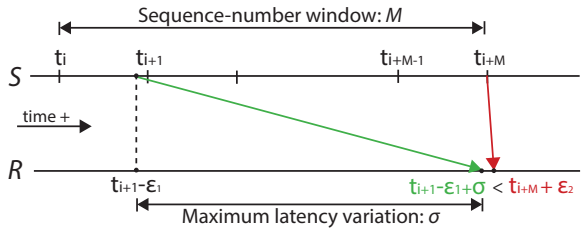

Figure 5: The last arriving packet with seqNo i must arrive before the first arriving packet with seqNo $i+M$.

Furthermore, to account for the worst case, we assume that R has the fastest clock, i.e., R thinks T has passed when in reality $0.95 \cdot T$ has passed, and S has the slowest clock, i.e., S thinks T has passed when in reality $1.05 \cdot T$ has passed. Thus, we set $\Delta = 0.1 \cdot T$.

Sequence-number window length (M). Equation 1 expresses the length of the seqNo window that a router maintains, as a function of the period that a single seqNo is used (T) and of the maximum latency variance (σ) between two packets.

The inequality is derived as follows: let t_i denote the time at which S starts using seqNo i. Since seqNos are updated every T, the elapsed time between the start of two seqNos is $t_i - t_j = (i-j)\cdot T$, for $i > j$. Then, consider a router that accepts packets in the window $[i, i+M-1]$. To ensure that a legitimate packet is not dropped due to packet reordering, the last packet with a seqNo of i should arrive at R before the first packet with a seqNo of $i+M$ (Figure 5). In the worst case, the last packet with seqNo i is sent at $t_{i+1} - \varepsilon_1$ and received at $t_{i+1} - \varepsilon_1 + \sigma$, where ε_1 is an arbitrarily small positive constant. The first packet with seqNo $i+M$ arrives at R no earlier than $t_{i+M} + \varepsilon_2$, where ε_2 is an arbitrarily small positive constant. Mathematically, the relation between the packet-arrival times can be described as $t_{i+1} - \varepsilon_1 + \sigma < t_{i+M} + \varepsilon_2$. By rearranging the inequality and using the fact $t_{i+M} - t_{i+1} = (M-1)\cdot T$, we obtain Equation 1.

Bloom-filter parameters (m, k, N, L). Equation 2 expresses the number N of required BFs as a function of the BF rotation interval L. Recall that R has to store a packet at least until the seqNo of the packet is no longer valid, which is a time period of length $M \cdot (T+\Delta)$.[3] Therefore the complete rotation of the circular BFs, which lasts for $N \cdot L$, should take longer than $M \cdot (T+\Delta)$, and this yields that $N \cdot L > M \cdot (T+\Delta)$. We obtain that $N > M \cdot (T+\Delta)/L + 1$ filters are required; the additional filter is necessary to store the incoming packets at the current time interval. We combine the formed inequality with Equation 1 and by setting $\Delta = 0.1 \cdot T$ we obtain Equation 2.

Equation 3 describes the size m of each BF as a function of the BF rotation interval L, the number N of BFs, the number k of necessary hash functions, and the BF's target false-positive rate (fp). Since an incoming packet is checked against all BFs, the overall target false-positive rate is $1-(1-fp)^N$. To determine the value for fp, we consider the average number of packets that a router receives in an interval L (which is $r \cdot L$, where r is the incoming packet rate). Using the BF equations, we get $fp = (1-e^{k \cdot x \cdot L/m})^k$ and by combining it with the equation for the size of a BF, we obtain Equation 3. The inequality indicates that any larger value for m yields a lower false-positive than fp.

The formulated optimization problem is an integer programming problem, which is known to be NP-hard [46]. Note that also the ro-

[3] In the worst case, a router does not receive seqNo updates from the source and self-increments the seqNo every $T+\Delta$.

Parameter	Value	Parameter	Value
T	10 ms	m	8 MB
r	14.88 Mpps	k	11
σ	100 ms	N	2
M	11	L	121 ms

Table 2: Software-router Implementation

tation interval L is an integer: a time period in a computing system is expressed as a multiple of some minimum supported time granularity (e.g., 1 ns); in practice, we will use values at the order of 1 ms (Section 5). In our context, we obtain multiple solutions to the problem by searching a constrained parameter space; for example, we constrain the size of the BF to be less than 20 MB, since ideally it should fit into a processor's cache. Our grid search is performed as follows: $10\ ms \leq L \leq 200\ ms$, $2 \leq N \leq 20$, $2 \leq k \leq 30$, $1\ MB \leq m \leq 20\ MB$.

Furthermore, the objective of the optimization problem changes depending on the implementation platform (e.g., software vs. hardware-based implementation). In Section 5.1, we adapt the optimization problem to a software router and show how a selection of carefully chosen parameters leads to an efficient implementation.

5. SOFTWARE PROTOTYPE

To demonstrate the practicality of our approach, we implement the proposed replay-suppression mechanism on a software router. Our evaluation focuses on the overhead of replay suppression and not other functionalities (e.g., source authentication or longest prefix matching). Thus, we do not consider a specific underlying network architecture, but we make the following generic assumptions:

- Every packet injected into the network by a host has a unique network-layer identifier. For example, the IP-ID field in IPv4 is implemented by most operating systems as a packet counter [33]. We use this identifier together with the immutable content of a packet to uniquely identify the packet and minimize the probability of a collision.
- A router can obtain the AS number (ASN) of the source-host for every packet. For example, certain network architectures express addresses as a $(ASN : hostID)$ tuple [16, 17]; or an IP forwarding information base (FIB) can be extended to include this information for every source-address prefix.

5.1 Implementation

The focus of our implementation is to optimize memory-access patterns. Since our solution is a memory-intensive application[4], forwarding performance depends mostly on cache efficiency, i.e., it depends on the memory footprint of the application and on the memory access patterns. Small data structures are more likely to fit in the cache and, thus, reduce the importance of the access pattern. However, in a software implementation the cache is shared with other processes and a small memory footprint does not guarantee optimal performance. Thus, we focus on minimizing cache misses.

To minimize cache misses, we use a blocked BF [47] instead of a standard BF. A blocked BF consists of multiple standard BFs (called blocks), each of which fits into the typical 64-byte cache line. For each element that is checked/inserted, the first hash value determines the block to be used and additional hash values determine which bits to check/set in the block. Thus, a blocked BF needs one cache miss for every operation in the worst case. This optimization comes at the cost of a larger memory footprint compared to a standard BF with the same false-positive rate.

[4]For each packet, $k \cdot N$ bits are accessed in the BFs; k bits for each one of the N BFs.

The next step to minimize cache misses is to minimize the number of blocked BFs. Recall from the protocol description (Section 4.2) that for every observed packet, we add it to the writeable BF and check for its presence in all other filters. Since blocked BFs may have one cache miss per checked/inserted element, we want to minimize the number of filters.

We solve the optimization problem (Section 4.3) with the objective of minimizing the number of BFs. To account for the worst case, we assume a packet rate of 14.88 Mpps, which is the theoretically maximum packet rate for a 10 GbE Network Interface Card. Also, we set a conservative value for the maximum latency variation σ to 100 ms, based on a recent latency-measurement study [48]. We target for an overall false-positive rate that is less than $5 \cdot 10^{-6}$, and we obtain multiple solutions that use $N = 2$ BFs (which is also the lowest possible value according to Equation 2). Specifically, we obtain solutions that have different filter sizes (m), different seqNo window lengths (M), and that rotate BFs at different time intervals (L). From these solutions we choose the one that has the smallest memory footprint (lowest m value), under the constraint that the filter size is a power of 2. This constraint provides a significant processing speedup, as heavily used computations are transformed to bitwise operations (e.g., modulo operations become bit-shifts). Table 2 summarizes all the parameters of our solution.

Furthermore, to check/insert elements in the BF, we need to obtain the pointers to the corresponding bits in the filter. To implement the keyed PRF (Section 4.2), we compute an AES based CBC-MAC over a *fixed length* of the first bytes of a packet, as a CBC-MAC is insecure for variable-length messages [49]. Also, from our analysis of CAIDA traces [50], we found that the first 48 bytes of a packet's content are sufficient to mitigate digest collisions; the same result has been reported by previous work [51]. We split the 16-byte output of the MAC into appropriately sized chunks so that the first chunk points to the 512-bit block and the remaining chunks point to the bits in the block.

The last required functionality is the FIB. The FIB holds for every AS S the seqNo SN_S^R and the count-down timer TTL_S; we decrement the TTL value every 1 ms.

Optimizations. We leverage the Data Plane Development Kit [52] and Intel AES-NI [53] to build our prototype, and we perform the following optimizations to the BF. To insert an element, we leverage 128-bit registers and an SSE *OR* instruction: we prepare the inserting element by setting the respective bits obtained from the MAC computation. Then, we set the required bits in the 512-bit block with four 128-bit SSE OR operations. To check for membership of an element, we use early exit, i.e., as soon as we discover an unset bit we know the element is not a duplicate. This results in better performance since false positives are low and it is very likely to discover unset bits early.

5.2 Evaluation

We evaluate the switching performance of our software router on a commodity server equipped with an Intel Xeon E5-2680 CPU (20 MB L3 cache), 32 GB DDR3 RAM, and a 10 GbE Network Interface Card (NIC). We dedicate only two cores of the CPU to perform all required processing: one core processes incoming packets, and the other core updates the TTL values and seqNos in the FIB.

We utilize Spirent SPR-N4U-220 as our packet generator to generate load on the router; the router processes the generated traffic and sends it back to the generator. We generate a FIB with 55k ASNs, and use random destination addresses to avoid spatiotemporal locality for FIB cache accesses.

First, we test the forwarding performance of one port for two packet sizes (64 and 128 bytes) and a representative mixture of In-

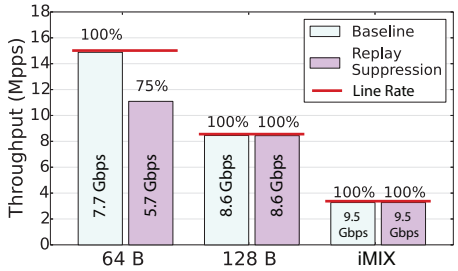

Figure 6: Forwarding performance for packet sizes of 64 and 128 bytes and for iMIX.

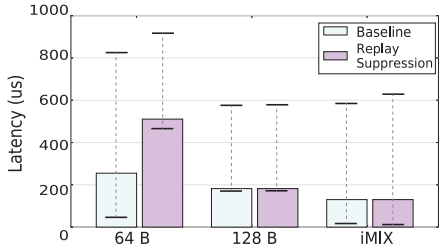

Figure 7: Average, minimum, and maximum packet latencies for packet sizes of 64 and 128 bytes and for iMIX.

ternet packet sizes (iMIX) [54]. Minimum-sized packets, 64 bytes, translate to the highest possible packet rate and are the worst case; we refer to the highest packet rate for each test case as the line-rate performance. The baseline for the experiments is the forwarding performance without any packet processing. Figure 6 shows the forwarding performance we obtain. The results show a 25% decrease for minimum-sized packets; and that for longer packet sizes, i.e., lower packet rates, optimal performance is achieved.

Next, we measure the latency overhead of our implementation (Figure 7). We observe a two-fold increase in average latency only for minimum-sized packets. The average latency and latency range is almost identical for the other two test cases.

We observe a performance degradation, both for throughput and latency, for minimum-sized packets. This performance degradation is attributed to the penalty of cache misses and the overhead of the MAC computation when the router is subjected to the maximum load. We emphasize that a 10 GbE link, fully utilized with 64-byte packets is far from a realistic workload. For a more realistic workload with iMIX, which has an average packet size of 417 bytes, our implementation saturates line-rate.

6. SECURITY CONSIDERATIONS

6.1 Deployment Location and Topology

We discuss certain security issues that depend on the location of routers that deploy replay suppression and on the network topology.

In Section 4, we mentioned that routers which deploy replay suppression are located at the borders of ASes. This deployment model raises certain security issues, which are mitigated if more routers inside an AS deploy the protocol.

Packet replays create an attack surface, which includes the path segments between the malicious router and the first deploying router that will drop the replayed packets. If deploying routers are located only at AS borders, then the attack surface is limited to a single AS. If more routers inside an AS deploy the protocol, then the attack surface is further reduced. For example, ASes could deploy more such routers near routing bottlenecks.

Furthermore, a malicious router can strategically replay packets even against deploying routers: since replay suppression is done by routers individually, without coordination among them, a packet that is sent to one deploying router can be successfully replayed (once) to another deploying router. However, the effect of such an attack is limited: the malicious router can replay a packet at most once to a router that performs replay suppression; additional replays after the first one will be suppressed.

6.2 Attacks on Bloom Filters

BF implementations are a common target for attackers [55]. We consider two types of attacks that could be launched against our protocol.

1. **Chosen-insertion Attack.** In a chosen-insertion attack, the adversary crafts packets that fill up the bits in the BF so that the false-positive rate becomes very high. Our protocol is resilient against the attack because we use a keyed PRF to compute the bit locations in the filter. Since the key is not known to the adversary and the output of the PRF is uniformly random, the adversary cannot set specific bits in the filter.
2. **Query-only Attack.** In the query-only attack, the adversary attempts to launch a DoS attack against the BF by querying items that take an abnormally long time to check. In our protocol, we focus on cache efficiency, so that either the BFs fit entirely in the cache or that checking for an item requires at most one cache miss.

6.3 Sequence-number Wrap-around

The use of per-interval seqNos makes wrap-arounds (i.e., restarting from zero) infrequent,[5] but wrap-arounds will eventually happen. In the event of a wrap-around, previously invalidated packets can be replayed because the seqNos become valid again. Furthermore, an adversary can replay previously seen packets that have higher seqNos than the one currently used by a source AS, so that the router would fast-forward the seqNo window for the source AS. As a result, the router would drop all legitimate packets that are sent by the source AS.

This problem is inherently solved by the underlying source authentication mechanism. In source-authentication schemes, source ASes periodically update their keys, so if the source AS updates the key before its seqNo wraps around, then the old packets will be invalid due to an authentication failure.

7. DISCUSSION

Hardware Implementation. In our software-router implementation, we used blocked BFs because the cache is shared with other processes.

In a hardware implementation, however, the optimization objective changes since a NIC can have a dedicated cache for the purpose of replay suppression. Standard bloom filters are a better option, because they are more space efficient than blocked bloom filters (for a given false-positive rate) and thus, can potentially fit into a dedicated cache. Table 3 summarizes the parameters of the optimization problem for a hardware-based implementation; the aggregate footprint of the application is less than 12 MB, with a false-positive rate of $9.85 \cdot 10^{-7}$.

Compliance to Sequence-number-update Interval. Recall that the only parameter that requires global agreement is the interval T at which ASes update their sequence numbers. We argue that ASes have no incentive to deviate from T. If AS S updates SN_S too fast, S may experience packet dropping due to packet reordering: packets with higher sequence numbers may arrive faster than packets with lower sequence numbers; this risk increases as T becomes smaller.

[5]For a 4-byte seqNo that is incremented every 10 ms, it takes about 497 days to wrap-around.

Parameter	Value	Parameter	Value
T	10 ms	m	4 MB
r	14.88 Mpps	k	11
σ	100 ms	N	3
M	11	L	61 ms

Table 3: Hardware-based Implementation

If S updates SN_S too slowly, S may experience packet dropping due to low seqNos: a router R may self-update its seqNo for S and the seqNos in the packets may fall out of the seqNo window.

Failure Recovery. Intermediate routers maintain seqNos and previously forwarded packets for all ASes. In an event of a failure (e.g., loss of power), a router may lose this information and thus, is unable to identify replayed packets after reboot. This is only a temporary situation: upon receiving a packet from a source AS, the router synchronizes its seqNo for the source, allowing it to filter any packets that fall out of the seqNo window, but not replayed packets with valid seqNos; however, after at most $M \cdot (T + \Delta)$ since the seqNo update, the router has fully recovered from the failure and can suppress all replayed packets.

A source AS (or a router) may fail as well. In case of a router failure in the source AS, the router asks a neighboring router within the same AS to determine the current seqNo as well as the time for next seqNo update. In case of a catastrophic failure of the entire AS, there are three choices. The source AS could use a sufficiently high seqNo so that routers in other ASes can synchronize to the new seqNo of the source AS. Or, the source AS can ask its neighboring ASes (or their neighboring routers while re-establishing BGP sessions) for the seqNo that it was using prior to failure. Alternately, as a last resort, a transit AS can erase state for the source AS, if it does not observe traffic from the source AS for a sufficiently large period of time; then, the transit AS uses the seqNo of the source AS when it observes again traffic from the source AS.

8. OTHER RELATED WORK

Section 3 describes the design space for end-to-end replay suppression. In this section, we describe some underspecified proposals that mention in-network replay suppression. We highlight that *all* related proposals do not satisfy the constraints for in-network deployment (Section 3.2).

Passport [17] proposes a combination of rapid re-keying and BFs. More specifically, a source domain and a transit domain use their shared secret to seed a hash chain that is used in a decreasing order. Then the hash value is used as a symmetric key to validate MACs for a short interval, and a BF stores the packets that were observed during the interval. To deal with packet reordering, two BFs are used, each of them storing packets for an interval of 5 seconds. The authors estimate a throughput of 781 Kpps with an SRAM requirement of 32 MB; we achieve 11.1 Mpps (over 1300% increase) with an SRAM requirement of 16 MB. To support a fully-saturated 10 GbE link with 64-byte packets, as in our evaluation, Passport would need more than 300 MB of storage.[6] Furthermore, the BFs are vulnerable to chosen-insertion attacks, since a simple hash function is used to compute the bit locations in the BFs. Thus, an attacker can force a router to drop all traffic.

Secure Network Attribution and Prioritization (SNAP) [56] recognizes the need for in-network replay detection to prevent replay of high-priority packets that will consume link capacity. Each router inserts a time-based serial number in every signed outgoing packet and every other router must verify the serial number; thus, routers have to keep per-router state. Furthermore, it is not specified how the effects of packet reordering are mitigated.

Furthermore, in-network replay suppression has been studied in a few other scenarios. Feng et al. devise a suppression mechanism for wireless networks that relies on digital signatures and achieves a throughput of 30 Kpps [57]. Replay attacks are also a notable concern for Named Data Networking (NDN)—a novel communication architecture for the Internet [58]. Compromised routers can capture and replay packets of content consumers at a later point in time. Proposed mechanisms are based on digital signatures and clock synchronization [59], but a concrete and practical mechanism is not specified. Although previous work suggests directions to mitigate replay attacks in the network, we provide the only practical mechanism that can sustain high throughput.

9. CONCLUSION

In this paper, we make the case for a seemingly counter-intuitive argument: replay suppression is becoming a critical functionality at the network layer. To demonstrate its importance, we present the router-reflection attack, in which compromised routers target a region of the Internet by flooding routing bottlenecks through packet replays. The attack is tenacious and pervasive: it is feasible even with source authentication, and there are hundreds or thousands of routers that can be compromised to carry out the attack.

We propose an in-network replay suppression protocol that is practical for today's hardware capabilities; our software-router prototype achieves line-rate performance for a fully saturated 10 Gbps link and for all but minimum-sized (64-byte) packets. The deployment of replay suppression at the network layer comes with further interesting implications: loops are inherently prevented, thus routers do not need to process the Time-To-Live field nor recompute the checksum in packet headers. In addition, it ensures that every bit in the network is attributable to its source.

10. ACKNOWLEDGMENTS

We would like to thank Pawel Szalachowski, Samuel Hitz, and the anonymous reviewers for their insightful feedback and suggestions. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement 617605; from the ICT R&D program of MSIP/IITP (No. B0717-17-0040, Development of self-certifying ID based trustworthy networking technology); and from NSF under Contract No. NSF CNS-0953600. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies of any other party. We also gratefully acknowledge support by ETH Zürich, the Zürich Information Security Center (ZISC), and Intel for their equipment donation that enabled the high-capacity experiments.

11. REFERENCES

[1] A. D. Birrell, "Secure Communication Using Remote Procedure Calls," *ACM Transactions on Computer Systems*, 1985.
[2] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)," RFC 4120, 2005.
[3] P. Thermos, "Examining Two Well-Known Attacks on VoIP," http://goo.gl/b4IEW3, Apr. 2006.
[4] R. Ludwig and M. Meyer, "The Eifel Detection Algorithm for TCP," RFC 3522, 2003.
[5] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security," RFC 4347, 2006.
[6] S. Kent, "IP Authentication Header," RFC 4302, 2005.
[7] S. Kent, "IP Encapsulating Security Payload (ESP)," RFC 4303, 2005.

[6]We note that Passport neglects important implementation details that affect performance (e.g., the hash function); thus, it is hard to provide a fair comparison. Our comparison favors Passport, as we neglect the effect of cache misses when we project the load of a fully utilized link to their scheme.

[8] S. Kent, "Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP)," RFC 4304, 2005.

[9] D. E. 3rd, "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)," RFC 4305, IETF, Dec. 2005.

[10] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, 1984.

[11] "Cisco Routers Compromised by Malicious Code Injection," http://goo.gl/oWBtF6, Sep. 2015.

[12] "Juniper ScreenOS Authenticatoin Backdoor," https://goo.gl/umV2gD, Dec. 2015.

[13] "Snowden: The NSA planted backdoors in Cisco products," http://goo.gl/xwdFW2, May 2015.

[14] R. M. Hinden, "Why Take Over the Hosts When You Can Take Over the Network," "http://bit.ly/2apRAxZ", Feb. 2014.

[15] "SDN Security Challenges in SDN Environments," http://bit.ly/2av8huG.

[16] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," in *Proc. of ACM SIGCOMM*, 2008.

[17] X. Liu, X. Yang, D. Wetherall, and T. Anderson, "Efficient and Secure Source Authentication with Packet Passports," in *Proc. of USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2006.

[18] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight Source Authentication and Path Validation," in *Proc. of ACM Conference on SIGCOMM*, 2014.

[19] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and Enforcing Network Paths with ICING," in *Proc. of the ACM Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2011.

[20] C. Pappas, R. M. Reischuk, and A. Perrig, "FAIR: Forwarding Accountability for Internet Reputability," in *Proc. of the IEEE International Conference on Network Protocols (ICNP)*, 2015.

[21] C. Basescu, R. M. Reischuk, P. Szalachowski, A. Perrig, Y. Zhang, H.-C. Hsiao, A. Kubota, and J. Urakawa, "SIBRA: Scalable Internet Bandwidth Reservation Architecture," in *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2016.

[22] Y. Go, E. Jeong, J. Won, Y. Kim, D. F. Kune, and K. Park, "Gaining Control of Cellular Traffic Accounting by Spurious TCP Retransmission," in *Proc. of the Annual Network and Distributed System Security Symposium NDSS*, 2014.

[23] M. S. Kang and V. D. Gligor, "Routing Bottlenecks in the Internet: Causes, Exploits, and Countermeasures," in *Proc. of the ACM Conference on Computer & Communications Security (CCS)*, 2014.

[24] M. S. Kang, S. B. Lee, and V. D. Gligor, "The Crossfire Attack," in *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2013.

[25] A. Studer and A. Perrig, "The Coremelt Attack," in *Proc. of the European Symposium on Research in Computer Security (ESORICS)*, 2009.

[26] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating Internet Bottlenecks: Algorithms, Measurements, and Implications," in *Proc. of ACM SIGCOMM*, 2004.

[27] CAIDA, "Observing Routing Asymmetry in Internet Traffic," https://goo.gl/uE4V3B.

[28] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, "On Routing Asymmetry in the Internet," in *Proc. of the IEEE Global Communications Conference (GLOBECOM)*, 2005.

[29] "CAIDA: Looking Glass API," http://goo.gl/AKQcd9.

[30] Cisco, "How Does Load Balancing Work?" http://goo.gl/SVbYM9.

[31] Juniper, "Configuring Per-Packet Load Balancing," http://goo.gl/ZO4LiS.

[32] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. Van Wesep, T. Anderson, and A. Krishnamurthy, "Reverse Traceroute," in *Proc. of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010.

[33] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP Topologies with Rocketfuel," in *Proc. of ACM SIGCOMM*, 2004.

[34] "RIPE Atlas," "http://atlas.ripe.net".

[35] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding Traceroute Anomalies with Paris Traceroute," in *Proc. of the ACM Conference on Internet Measurement (IMC)*, 2006.

[36] K. Keys, Y. Hyun, M. Luckie, and K. Claffy, "Internet-scale IPv4 Alias Resolution with MIDAR," *IEEE/ACM Transactions on Networking*, 2013.

[37] K. Zetter, "NSA Laughs at PCs, Prefers Hacking Routers and Switches," http://bit.ly/2awxsN8, Apr. 2013.

[38] Cisco, "Cisco IOS Software Integrity Assurance," http://bit.ly/2ab76RE.

[39] T. Lee and B. Hau, "The New Route to Persistence: Compromised Routers in the Wild," http://bit.ly/1ObMm7u, Sep. 2015.

[40] P. Porras, "Towards a More Secure SDN Control Layer - SRI International's View," http://bit.ly/2ax0ERr, Oct. 2013.

[41] G. Pickett, "Abusing Software Defined Networks," http://ubm.io/1sT9QTD, Oct. 2014.

[42] S. Luan and V. D. Gligor, "On Replay Detection in Distributed Systems," in *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 1990.

[43] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and Adoptable Source Authentication," in *Proc. of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2008.

[44] ARIN, "Resource Public Key Infrastructure."

[45] H. Marouani and M. R. Dagenais, "Comparing High Resolution Timestamps in Computer Clusters," in *Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering*, 2005.

[46] C. H. Papadimitriou, "On the Complexity of Integer Programming," *Journal of the ACM (JACM)*, 1981.

[47] F. Putze, P. Sanders, and J. Singler, "Cache-, Hash- and Space-efficient Bloom Filters," *Journal of Experimental Algorithmics (JEA)*, 2009.

[48] R. Durairajan, S. K. Mani, J. Sommers, and P. Barford, "Time's Forgotten: Using NTP to Understand Internet Latency," in *Proc. of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2015.

[49] M. Bellare, J. Kilian, and P. Rogaway, "The Security of the Cipher Block Chaining Message Authentication Code," *Journal of Computer and System Science*, 2001.

[50] "The CAIDA UCSD Anonymized Internet Traces 2015-050615," http://goo.gl/WmItAH.

[51] N. G. Duffield and M. Grossglauser, "Trajectory Sampling for Direct Traffic Observation," *IEEE/ACM Transactions on Networking*, 2001.

[52] "Data Plane Development Kit," "http://dpdk.org".

[53] S. Gueron, "Intel Advanced Encryption Standard (AES) New Instruction Set," https://goo.gl/of08Dg, March 2010.

[54] A. Morton, "IMIX Genome: Specification of Variable Packet Sizes for Additional Testing," RFC 6985, 2013.

[55] T. Gerbet, A. Kumar, and C. Lauradoux, "The Power of Evil Choices in Bloom Filters," INRIA Grenoble, Research Report RR-8627, 2014.

[56] G. D. Troxel and L. P. Ma, "Secure Network Attribution and Prioritization: A Coordinated Architecture for Critical Infrastructure," in *Proc. of the IEEE Military Communications Conference (MILCOM)*, 2013.

[57] Z. Feng, J. Ning, I. Broustis, K. Pelechrinis, S. V. Krishnamurthy, and M. Faloutsos, "Coping with Packet Replay Attacks in Wireless Networks," in *Proc. of the IEEE Conference on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON)*, 2011.

[58] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proc. of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009.

[59] A. Afanasyev, "Addressing operational challenges in named data networking through ndns distributed database," Ph.D. dissertation, Citeseer, 2013.