Dependable Connection Setup for Network Capabilities

Soo Bum Lee Virgil D. Gligor Adrian Perrig
CyLab, Carnegie Mellon University
Email: {soobum, gligor, perrig}@cmu.edu

Abstract

Network-layer capabilities offer strong protection against link flooding by authorizing individual flows with unforgeable credentials (i.e., capabilities). However, the capability-setup channel is vulnerable to flooding attacks that prevent legitimate clients from acquiring capabilities; i.e., in Denial of Capability (DoC) attacks. Based on the observation that the distribution of attack sources in the current Internet is highly non-uniform, we provide a router-level scheme that confines the effects of DoC attacks to specified locales or neighborhoods (e.g., one or more administrative domains of the Internet). Our scheme provides precise access guarantees for capability schemes, even in the face of flooding attacks. The effectiveness of our scheme is evaluated by ns2 simulations under different attack scenarios.

1. Introduction

Current service-flooding attacks rely on a large number of compromised machines that are organized as a "bot" network. Typical defense mechanisms that attempt to provide service-access guarantees despite such attacks assume absence of flooding in the underlying network links. Yet, a large-scale attack (e.g., a "botnet" with millions of "bots") can flood any chosen link in the Internet. In particular, defense mechanisms deployed at links near or at a network edge (e.g., Firewalls, IDSs) can be easily overwhelmed by such attacks. Worse yet, legitimate-looking attack packets can evade most of traditional techniques for handling address spoofing attacks at the network layer (e.g., IP tracebacks [13, 15], ingress filtering [8]).

Capability-based solutions, whereby distinct packet flows are separately authorized through capabilities obtained before flow initiation [3, 20, 21], provide congested routers with an effective way to prioritize legitimate flows and filter out unwanted traffic. Though promising, these solutions are still vulnerable to flooding attacks targeting the *capability-setup channel*, known as the Denial of Capability (DoC) attacks [4]. These attacks are possible because the initial capability-request packets are treated as best-effort packets, as opposed to the subsequent high-priority packets that carry capabilities. If DoC attacks cannot be countered, flow authorization via network-layer capabilities becomes impossible, and all access guarantees become meaningless at congested routers.

Previous solutions that attempt to protect capability requests from flooding attacks (e.g., mechanisms based on aggregate request rates [21] or on proof of work [12]), though useful, are insufficient to provide dependable link-access guarantees for legitimate capability requests. For example, a fair-queueing mechanism, which fairly allocates buffer space to flow aggregates based on a router's confidence in precise identification of traffic origin [21], fails to provide *any* guarantee of link-access (viz., Section 7.1). Mechanisms based on proof of work (e.g., Portcullis [12]) provide only *weak* access guarantees during flooding attacks as they are (at best linearly) dependent on the number of global attack sources; e.g., a large number of bots could still flood a chosen link despite such guarantees. These previous schemes achieve relatively weak guarantees because they assume that attack sources are uniformly distributed in the network.

We observe, however, that malicious hosts, or bots are clustered: some domains include sufficiently strong security mechanisms that enable them to counter or deter contamination; others are easily contaminated by bots. [1] Non-uniform distribution of attack sources actually enables us to achieve stronger guarantees. To be meaningful, these guarantees have to be independent of the number of attack sources (i.e., the size of a global botnet). In the worst case, they can only depend on attack sources in *defined locales* or neighborhoods (e.g., an administrative domain or a set of domains in the Internet). As a consequence, competing requests for a capability to a congested link that originate outside a contaminated locale should be unaffected, or only minimally affected, by a flooding attack, and should receive strong access guarantees. In contrast, initial capability requests originating from bot-contaminated locales should receive weaker access guarantees, namely guarantees that depend only on the number of bots in the contaminated locale (but not on *all* bots of a multi-domain attack network). In short, our no-

[1]Non-uniform distribution of attack sources is evident in a variety of worm propagation models [5, 16], evolutionary features of previous worms such as CodeRed I/II, Nimda and Slammer.

tion of dependable access to a flooded link provides *differential guarantees* for the capability setup channel. Differential access guarantees are desirable because they provide incentives for employing host security measures within administrative domains that prevent botnet (and other malware) contamination. In exchange, uncontaminated domains receive precise guarantees of link access for the *capability setup channel*, which support meaningful network-link and, ultimately, service-access guarantees.

Our scheme relies on three basic mechanisms. First, we define a new *path identification* mechanism that provides an unforgeable domain identifier to individual packets, and enables remote routers to identify a packet's domain of origin. Second, we define a *dynamic virtual queueing* mechanism that guarantees a minimum number of router buffer slots to domains originating flows through a router, which in effect, guarantees link access to those domains. Finally, we employ a *path aggregation* mechanism that optimizes router bandwidth allocation for legitimate capability requests based on domain contamination.

2. Background and Related Work

Lack of source address authenticity in the Internet Protocol (IP) enables attackers to forge the source addresses, and hence complicates/prevents address-based accounting during link flooding attacks. As a way to add authenticity to individual packets, capability solutions [3,20,21] have been proposed. Generally, a network-layer capability protocol requires a handshake between a client and a server, and during that phase, routers on the forwarding path collectively issue a connection capability; i.e., a series of router capabilities on the path. A router's capability, which is generated by hashing the source and destination IP address with the router's secret key, is cryptographically secure against forgeries since the router key is unavailable to an adversary.

However, the capability request protocol is still vulnerable to flooding (DoC) attacks [4]. That is, flooding with capability requests, which cannot be prioritized, successfully denies a legitimate access to a congested link. Portcullis [12] proposes a puzzle-based mechanism that provides a guaranteed link access during a flooding (DoC) attack. Though useful, the guarantee is linearly dependent on the number of bots, which can be substantial (e.g., the size of a botnet easily exceeds 1 million bots [1]). Alternatively, TVA's implementation of fair queueing on incoming traffic paths (i.e., hierarchical fair queueing) [21], which equally assigns queues to directly connected links and splits the queues recursively for distant links, places legitimate accesses of remote domains at a significant disadvantage since it provides fair service to the same level of queues (i.e., sub-queues split from a queue). More sophisticated application-layer solutions (e.g., CAPTCHA [18]) that attempt to distinguish between human- and machine-initiated traffic to prevent flooding attacks are impractical at the network-link level.

Attempts to block suspicious traffic upstream of a congested router by installing filters close to, or at, the domains originating attacks could protect legitimate flows that are independent of attacks. To be effective, cooperative filtering would require incentives that scale with the number of participating domains – a tall order since it depends on the attack itself. Furthermore, with only local information (the traffic rate of incoming links), a router cannot easily identify the links (or upstream links) that are responsible for the congestion; and even if such information is available, an adversary can launch a timed attack where different groups of zombies/bots issue targeted requests by exploiting the time delay required for installing and releasing filters at upstream routers (e.g., on-off and rolling attacks).

3. Design Overview

In this section, we present an overview of our defense scheme by describing the basic mechanisms.

3.1. Threat

The main threat we deal with in this work is a link flooding attack on the capability-setup channel, where attack sources collaboratively exhaust the link bandwidth allocated for connection establishment. We assume that both hosts and routers can be compromised and send/forward attack traffic. Compromised hosts are able to both flood a target link with capability request packets and disturb the path identification mechanism at a remote router by manipulating the header reserved for that purpose (viz., Section 4.1). Compromised routers can disturb path identification by either forwarding packets that contain false path-markings or adding invalid path-markings to the packets they forward.

3.2. Path Identification

In this work, we consider routers that mark packets with path information. These path-markings create an unspoofable *origin* identifier because they cannot be controlled by end-hosts.[2] In addition, path-markings enable remote routers to construct a traffic tree. The domain connectivity revealed in the traffic tree helps identify the distribution of attack sources in specified locales to which bandwidth allocation will be restricted (viz., Section 6).

The basic concept of route construction is similar to that of previous schemes [19,21], yet we use a packet's AS (Autonomous System) path as a domain identifier for several reasons. First, a packet's AS path, which is primarily determined by the number of AS hops (AS path length) to the destination in the inter-domain routing protocol (e.g., BGP-4), is more stable than the routing path within an AS that

[2]IP source routing may allow a client to select a path to a destination. However, strict and loose source routing are usually blocked at routers to avoid the associated processing overhead.

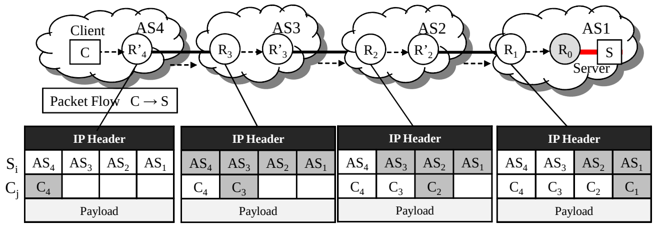

Figure 1: Path Identifier. R'_4 is the egress router of AS4 and R_3, R_2, R_1 are the ingress routers of AS3, AS2, AS1 respectively. R'_4 writes the path-identifer to the packet heading to server S in AS1, and ingress routers on the path validate the markings. C_j is the capability issued by R_j. Each ingress router can validate the shaded part of the markings.

may frequently change during flooding attacks due to link state changes (e.g., link failure). We use the AS path of a packet as a persistent domain identifier. Second, a packet's AS path can be constructed by the egress router of the source domain since the router contains the AS path information of destination addresses in its routing table. This source-constructible domain identifier eliminates deployment issues that plagued previous path-marking schemes especially in the Internet core, and hence enables independent adoption of the marking scheme at the Internet border (e.g., provider/stub domains). We envision that prioritizing requests originating from path-marking domains would encourage early adoption of the marking scheme.

We define a packet's AS path to its destination as the *path-identifier* of the packet, and present it in the order of markings: from the origin to the destination. Thus, as illustrated in Fig. 1, the path-identifier seen at a congested router in AS_1 is $\{AS_4, AS_3, AS_2, AS_1\}$. We implement this path-identifier in a shim header so that only upgraded routers interpret it. Throughout this paper, we denote the path-identifier whose markings start with AS_i by S_i and the BGP speaker of AS_i by R_i. In Section 4, we present a mechanism that protects path-identifiers from potential attacks.

3.3. Link Access Guarantees

In defending against DoC attacks, our goal is to provide precise guarantees of link access to capability requests, where the guarantees are provided in a domain basis to confine the effects of attacks within the domains from which they originate. This goal is achieved by a new fair queueing mechanism, and the guarantees provided by the queueing mechanism are optimized to favor the requests from uncontaminated domains via a path aggregation mechanism.

3.3.1 Fair Queueing Revisited

The choice of a fair queueing scheme for link-access guarantees is intended to maximize service on the legitimate capability requests. Fair queueing schemes, if they can assign separate queues to individual path-identifiers, could provide fair bandwidth to the path-identifiers without link under-utilization (which could occur whenever strict bandwidth reservation is made to individual path-identifiers). However, when the spatio-temporal dynamics of domains contributing to congestion (e.g., time-varying patterns of domain traffic) are considered, such queue assignment in a limited buffer is a challenging problem. For example, for a fixed buffer size, under-provisioning of the number of queues in a specific time period may fail to provide link-access guarantees to path-identifiers due to potential queue collisions among different path-identifiers. In contrast, over-provisioning of it would decrease the length of individual queues, hence weaken the guarantees (viz., Section 5). Thus, we aim to design a fair queueing scheme that assigns a unique queue to each path-identifier and adjusts the individual queue lengths to fit the buffer size for link-access guarantees and their enhancement – a desired goal.

While a variety of traditional fair queueing schemes focus on the bandwidth fairness of flows in different queues that contain various sizes of packets, the Stochastic Fair Queueing (SFQ) scheme [11] offers queue length fairness via a *buffer stealing* mechanism, whereby a packet that finds a full buffer on its arrival would steal a buffer-slot from the longest queue. We note that the *fixed size* capability request packet would eliminate the intrinsic bandwidth unfairness of SFQ in the presence of different packet sizes [14]. Based on the buffer-stealing idea, we improve SFQ in two respects. First, we avoid queue collisions among path-identifiers that are allowed but fairly distributed via stochastic queue assignment in SFQ. Second, we make queue management operations (e.g., queue assignment and buffer-slot preemption) scalable and efficient to easily adapt our scheme to diverse operating environments (e.g., link capacity, the number of required queues). Those improvements are made via a dynamic virtual queueing mechanism presented in Section 5.

3.3.2 Path Aggregation

As more domains are contaminated by attack sources, link-access guarantees provided by our queueing scheme become weak as both available link bandwidth and buffer-slots to each path-identifier decrease. This undesirable dependency of guarantees on attack dispersion is unavoidable as long as all path-identifiers are *equally* treated. Protecting requests of uncontaminated domains essentially needs a differential treatment of path-identifiers based on the proportion of legitimate requests they deliver. Though the legitimacy of individual capability requests cannot be validated, the proportion of legitimate requests in a set of requests can be estimated by a couple of flow conformance tests, which consist of (1) a test on *bandwidth conformance* that represents the aggressiveness of requests and (2) a test on *protocol conformance* that indicates the legitimacy of authorized flows in various respects (viz., Section 6.1).

Conformance tests performed on each path-identifier enables differential assignment of bandwidth to path-identifiers that maximizes service to legitimate requests at the flooded link. Yet, in the presence of a large number of attack domains, such assignment cannot easily be made, nor can it tolerate imprecise measurement of domain contamination. Instead, we aggregate the path-identifiers of a highly contaminated locale and assign a new path-identifier to them. This, in effect, limits both available bandwidth and buffer space for those path-identifiers. We define this path aggregation problem as a constrained optimization problem and provide an efficient solution in Section 6.3.

4. Path Identification

In this section, we start with the basic path identification mechanism, and then enhance the mechanism with additional security features.

The basic path identification mechanism works as follows. When the egress router of a domain (i.e., the BGP speaker) forwards a packet that originates from its domain, it writes the path-identifier (i.e., the AS path to the destination) in the packet's header. AS ingress routers of the packet forwarding path validate the authenticity of a fraction of this path-identifier starting with the upstream AS that forwarded the packet and ending with the destination AS as shown in Fig. 1. Whenever AS ingress routers receive a non-marked packet, they write their own path-markings: the AS path from their upstream AS to the destination AS.

As remote domains can validate only a part of path-markings, attack sources in unprotected (non-marking) domains may spoof path-identifiers unless the marking scheme (which includes the verification function) is sufficiently deployed. Even under wide deployment of the marking scheme, the authenticity of path-identifiers verified at a domain cannot be delegated to the downstream domains without a strong trust relationship established between those domains. This makes any manipulation of path-identifiers by compromised routers undetectable at remote routers. To protect path-identifiers from potential attacks (e.g., spoofing and replay attacks), we present a *secure* path identification mechanism below.

4.1. Unspoofable path-identifier

We first introduce potential attacks that disturb path-identification at remote routers and present our defense mechanism against those attacks.

Let $\{AS_n, \ldots, AS_2, AS_1\}$ be the path-identifier seen at the congested router, and let $*$ and $\#$ be any valid and forged sequence of markings respectively. Then, both compromised sources in unprotected domains and compromised routers in AS_k can forge a path-identifier as $\{\#, AS_i, *, AS_1\}$, if the domains up to AS_i are unprotected.

In principle, a router can authenticate its path-markings by adding a digital signature to the path-markings. However, adding a digital signature in every packet would impose significant computational overhead for both its generation and verification. Moreover, a per-packet signature, if employed,

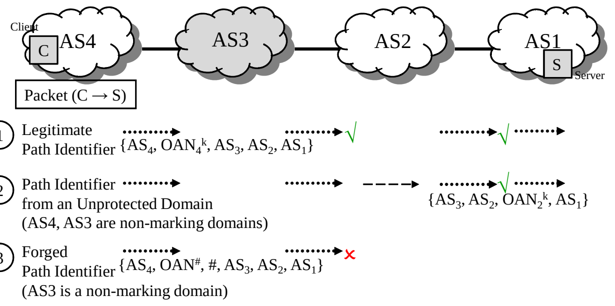
Client
C
AS4
AS3
AS2
AS1
S
Server
Packet (C → S)
1 Legitimate Path Identifier {AS4, OAN4k, AS3, AS2, AS1}
2 Path Identifier from an Unprotected Domain (AS4, AS3 are non-marking domains)
{AS3, AS2, OAN2k, AS1}
3 Forged Path Identifier {AS4, OAN#, #, AS3, AS2, AS1}
(AS3 is a non-marking domain)

Figure 2: Path-identifier Authentication. ① Path-identifier written at the packet's origin (AS4) can be validated at any domain (AS2, AS1) in the presence of a non-marking domain(s) (AS3) on the packet's forwarding path. ② If the origin AS does not participate in path-marking, the first participant (AS2) writes its markings and adds the incoming AS number (AS3) to distinguish the packets it forwards from the ones originating from it. ③ An invalid ASN-OAN pair (denoted by $\#$) can be detected and filtered.

could be exploited by attackers to exhaust routers' computational resources (e.g., by flooding small-size packets). Instead, we present an efficient path-identifier authentication mechanism, where each domain pre-distributes its domain-authenticator and uses it to authenticate its path-markings. One fundamental assumption for implementing this mechanism is that any protected AS has a public-private key pair certified by a trusted certificate authority (e.g., ICANN).

4.1.1 Authenticator Distribution

When a BGP speaker advertises an address prefix that belongs to its domain, the BGP speaker adds an origin authentication number (OAN), which is unique in its domain and is digitally signed with the domain's private-key, to its route advertisement. All BGP routers that receive this route advertisement validate the OAN with the origin's public-key and hold the authenticated ASN (AS Number)-OAN pair for later path-identifier authentication. Since the number of ASN-OAN pairs is at most 65,535[3], the space requirement for this validation is bounded, i.e., 262KB for 4-Byte OANs.

4.1.2 Origin Authentication

The BGP speaker of a packet's domain of origin writes its ASN-OAN pair followed by the AS path to the destination in the path-identifier header. Fig. 2 illustrates the cases for origin authentication under different deployment scenarios of the marking scheme. Whenever no path-identifier is present in a packet, the ingress router of a marking AS constructs path-markings with its own ASN-OAN pair (viz., ② in Fig. 2). On receiving path-identifiers constructed as such, the ingress routers of downstream domains validate the origin's OAN and the partial AS path as discussed above.[4] Thus, invalid path-markings can be identified even in the presence of

[3]As of 2010, the number of advertised ASNs is about 30,000 out of 65,535 (16-bit) possible ASNs [2]

[4]For path validation, routers need to keep AS path information (from next hop to the destination AS) in their forwarding table (i.e., FIB). However, this would not require much space since the average number of ASes a packet traverses from its origin to destination is four.

consecutive non-marking ASes on the paths and be filtered on the way to, or at the destination AS.

Meanwhile, a compromised router in AS_i can still forge two types of valid path-identifiers such as $\{AS_i, OAN_i^k, *\}$ and $\{\#, AS_i, OAN_i^k, *\}$. However, their effects can be limited to at most those of two path-identifiers by discarding the non-authenticated prefixes of path-identifiers.

4.2. Preventing Replay Attacks

Under partial deployment of our path-marking scheme, attack sources in unprotected domains may forge path-identifiers ending with authenticated ASN-OAN pairs (since ASN-OAN pairs are not confidential to end-hosts) and use them in flooding a target link. Such replay attacks would significantly affect the requests of protected domains.

Path-marking routers counter replay attacks via fast OAN renewals, which are efficiently implemented using a reverse hash chain [12]. Let OAN$_i^0$ be the initial OAN of AS_i. AS_i constructs a hash chain of OANs by repeatedly hashing OAN$_i^0$ with a cryptographic hash function (i.e., OAN$_i^k$ = Hash(OAN$_i^{k-1}||AS_i||k-1$) for $1 \leqslant k \leqslant M$), and distributes OAN$_i^M$ when advertising a route. We engage AS_i and $k-1$ in generating OAN to produce distinct OAN sequences for different ASes and initial OANs respectively. A BGP speaker uses OAN$_i^k$ during a predefined interval; and changes it to OAN$_i^{k-1}$ in the next interval. Hence, without breaking the hash function, an attacker cannot construct the valid sequence of OAN$_i^k$s to be used. A (ingress) router can authenticate OAN$_i^k$ by computing Hash(OAN$_i^k||AS_i||k$) and comparing it with OAN$_i^{k+1}$. This OAN authentication is performed only once for every OAN renewal. Once OAN$_i^k$ is used, OAN$_i^{k+1}$ is invalidated. Note that if the OAN renewal period is less than the time required for replaying OANs, replay attacks will be effectively prevented. The length of a OAN hash chain (M) is determined in consideration of the OAN renewal period to avoid frequent OAN distribution. For example, if a 20-bit sequence number (M $\approx$ 1 million) and 500ms OAN renewal period are used, a domain needs to advertise its OAN once in every six days. We also note that routers in different domains need not be time-synchronized as an OAN carries its sequence number that is specific to the domain.

5. Dynamic Virtual Queueing

In this section, we describe a dynamic virtual queueing mechanism for link-access guarantees on path-identifiers. Our dynamic virtual queueing mechanism is designed to assign a separate queue to active path-identifiers and provide queue length fairness to the path-identifiers in a *min-max* manner. For these purposes, a router manages virtual queues rather than physically separate queues, that are distinguished by the path-identifier (S_i), its count at time t ($N_{S_i}(t)$) and packet location (memory address) (A_{S_i}) in the buffer; i.e., $(S_i, N_{S_i}(t), A_{S_i})$. Given these tuples and the buffer size L_Q, queue-length fairness on path-identifiers ($\min\max_{S_i \in \mathcal{S}} N_{S_i}(t)$ for $\sum_{S_i \in \mathcal{S}} N_{S_i}(t) = L_Q$) can be described by the following buffer-slot preemption policy. If a packet finds the buffer full on its arrival, it preempts a buffer-slot from the longest virtual queue. If the arrived packet belongs to the longest virtual queue, or its preemption produces another longest virtual queue, it would be dropped. This preemption policy ensures guaranteed buffer-slots to each path-identifier if the number of buffered path-identifiers is bounded. We assume that the number of buffered path-identifiers can be statistically or deterministically bounded by $|\mathcal{S}|_{max}$ at a router (i.e., the minimum bandwidth to a legitimate path-identifier can be determined).

5.1. Implementing Buffer-slot Preemption

For efficient and scalable accounting of virtual queue lengths, we use a new Counting Bloom Filter (CBF) that holds the number of buffer-slots occupied by path-identifiers and provides lookup, add and remove operations in $O(1)$ time (a modified version of CBF [6]). CBF consists of m counter arrays of size 2^b $(a_1, a_2, \ldots, a_m)$ and m hash functions of b-bit output $(H_1, H_2, \ldots, H_m)$, where a_i is associated with H_i. For an input to CBF, each hash function maps its output to the corresponding array position; e.g., $a_i[H_i(S_1)]$ corresponds to the input S_1 for $1 \leqslant i \leqslant m$.

Path-identifier accounting in CBF works as follows. All array values are initialized to zero. When a packet is added to the buffer, its path-identifier is fed into CBF. Then, CBF locates m array positions for the path-identifier, and increases the corresponding array values. The same applies to a packet removal from the buffer, yet the counter values are decreased. In this scheme, the limited hash output size (i.e., 2^b) could cause hash-output collisions among path-identifiers. Such collisions would make corresponding array values increased by multiple path-identifiers, hence corrupted. However, unless all of the array values associated with S_i are corrupted, we can compute the count of buffered S_i's by taking the minimum of the array values; i.e., $\min\{a_1[H_1(S_i)], a_2[H_2(S_i)], ..., a_m[H_m(S_i)]\}$. Since the probability that all m array values of a path-identifier are corrupted is $(1-(1-(1/2^b))^{|\mathcal{S}|})^m$ for $|\mathcal{S}|$ buffered path-identifiers [7], we can make the probability negligible by increasing the array size (2^b) or the number of arrays (m).

Path-identifiers that occupy more buffer slots than the guaranteed amount (i.e., $\lfloor \frac{L_Q}{|\mathcal{S}|} \rfloor$) should be kept track of for possible preemption. To this end, a router maintains a table, named Path-Identifier Record (PIR), that holds *over-buffered* path-identifiers, their counts and corresponding packet locations. In PIR, a path-identifier is stored as the concatenation of m hash outputs of it, defined as "*path-signature*." This enables fast buffer-slot preemption because the preempted packet's path-signature would directly locate array values that need to be decreased in CBF.

5.2. Probabilistic Guarantees

If the packet arrivals of path-identifier S_i are modeled as a Poisson process and k buffer-slots are allocated to S_i, the probabilistic lower bound of S_i's link access (denoted by $\mathcal{G}(|\mathcal{S}|, k, S_i)$) is provided as follows.

$$\mathcal{G}(|\mathcal{S}|, k, S_i) = \tag{5.1}$$
$$\begin{cases} \sum_{j=0}^{k-1} \frac{(k\rho_{S_i})^j}{j!} e^{-k\rho_{S_i}} & \rho_{S_i} < 1 \\ \frac{1}{\rho_{S_i}}(1-\mathcal{G}_{\mathcal{L}})(1-\sum_{j=k}^{\infty} \frac{(k\rho_{S_i})^j}{j!} e^{-k\rho_{S_i}} \binom{j}{k-1} \mathcal{G}_{\mathcal{L}}^{k-1}) & \rho_{S_i} \geqslant 1 \end{cases}$$

where λ_{S_i} is the request rate of S_i, $\rho_{S_i} = \frac{\lambda_{S_i}|\mathcal{S}|}{C_R}$ is the bandwidth utilization of S_i, and $\mathcal{G}_{\mathcal{L}} = \sum_{j=0}^{k-1} \frac{(k\rho_{S_i})^j}{j!} e^{-k\rho_{S_i}}$.

We justify the Poisson arrival model of capability requests with two reasons: (1) during the *short* interval that the guarantees are defined (i.e., the maximum queueing delay of a router Δ_Q), the capability requests by different clients can be assumed independent; and (2) a single capability can be used for multiple *correlated* sessions that need to be established for most Web applications. Under this model, if $\rho_{S_i} < 1$, an arrival of S_i is guaranteed to be serviced if less than k arrivals of S_i has occurred in Δ_Q. If $\rho_{S_i} \geqslant 1$, an arrival of S_i is guaranteed to be serviced only if its queue length is less than k. Thus, Eq. (5.1) can be easily proved. The probabilistic guarantee of S_i's link-access is provided by setting $|\mathcal{S}| = |\mathcal{S}|_{max}$ and $k = \lfloor \frac{L_Q}{|\mathcal{S}|_{max}} \rfloor$.

5.3. Resource Requirements

5.3.1 Request Packet Buffer

A large buffer (L_Q) for capability request packets is preferable since it would not only improve the guarantees (viz., Eq. (5.1)) but also handle the requests from spontaneously created, short-lived paths. However, the size of the buffer should be bounded in consideration of the maximum allowed queueing delay to avoid unnecessary retries at flow sources. For example, if we assume 0.25 second maximum queueing delay and 128B[5] request packet size, for a 2.5 Gbps link [6], a router requires 4.0 MB buffer (when 5% of link bandwidth is allocated for capability requests [21]), and with which it can provide 8 guaranteed buffer slots up to 3.75K path-identifiers.

5.3.2 Path-Identifier Accounting

The memory requirement for CBF is determined by a target false-positive ratio. The false positive ratio of a CBF is determined by $\left(1-(1-\frac{1}{2^b})^{|\mathcal{S}|}\right)^m \approx \left(1-e^{-\frac{|\mathcal{S}|}{2^b}}\right)^m = \left(1-e^{-\frac{L_Q}{k\cdot 2^b}}\right)^m$ since $L_Q = k \cdot |\mathcal{S}|$. Hence, for a desired false positive ratio, the size of each counter array in CBF, which is same as the size of hash output (2^b), is linear with the buffer size (i.e., $\Theta(L_Q)$). For example, a CBF with 8 hash functions of 14-bit outputs would require 8×2^{14} (hash outputs) $\times 2^8$ (counter) = 131KB memory space while producing a reasonably low false positive ratio ($3.07 \times 10^{-4}\%$) in the presence of 3.75K path-identifiers.

PIR holds the path-identifiers whose count exceeds $\lfloor \frac{L_Q}{|\mathcal{S}|} \rfloor$ for possible preemption. Hence, the memory requirement is bounded by $L_Q/(k+1)\times$ (16B (path-signature) + 4B (address pointer)) (e.g., 60KB for the above example), since the number of path-signatures in PIR has its maximum when all path-identifiers have $k+1$ packets in the buffer. Hence, the memory requirement for both CBF and PIR is $\Theta(L_Q)$.

6. Path Aggregation

In this section, we first describe a mechanism for estimating the proportion of legitimate requests of individual path-identifiers, and then, a path-identifier aggregation mechanism that maximizes the *goodput ratio*, defined as the proportion of legitimate requests in all *serviced* requests, at a congested link. Aggregating path-identifiers produces an optimal traffic tree to which applying our queueing mechanism maximizes goodput ratio at the congested link.

6.1. Goodput Estimation

In absence of any other useful information regarding the origin of attack sources and the path-identifiers assigned to them, the request rate of path-identifier S_i (λ_{S_i}) can be used as a unique measure for estimating the goodput ratio of S_i. We define the *bandwidth conformance* of path-identifier S_i as $\min\{1, \frac{C_R}{\lambda_{S_i}|\mathcal{S}|_{max}}\}$ to represent how the request rate of S_i conforms to the assigned bandwidth to it, and denote it by $\mathcal{E}^{\mathcal{B}}_{R_i}$, i.e., $\mathcal{E}^{\mathcal{B}}_{R_i} = \min\{1, \frac{C_R}{\lambda_{S_i}|\mathcal{S}|_{max}}\}$ (recall that S_i is assigned to all packets originating from R_i).

Additionally, we estimate domain contamination more accurately by identifying the following attack flows.

Unauthorized flows: A capability issued by a router during the connection establishment phase of a flow must be used at least once for actual data transmission unless it is denied afterward by application services, firewalls or IDSs. Thus, the proportion of unused capabilities could effectively measure domain contamination as it reflects the strong flow authorization results applied at the network ends.

High-rate flows: Flows that send high-rate traffic using valid capabilities would exhibit high packet-drop rates as indicated in [10]. Hence, if a router implements per-domain bandwidth control,[7] high-rate attack flows within a domain can be identified by capability drop rates.

[5]We reserve 88B shim header: 40B for path-identifiers (up to 10 AS markings), 8B for an origin authenticator and 40B for 5 capabilities.

[6]2.5Gbps (OC–48) links are widely used for ISP's backbone links.

[7]Flows in different domains could exhibit different drop rates due to different RTTs.

High-fanout sources: If sources are allowed to establish an unlimited number of connections with other destinations through the congested link, they can deplete link's bandwidth with a large number of legitimate-looking flows [17]. This insidious attack will be prevented if a router limits the number of per-source capabilities as follows.

Let $C_{f_{s,d}}$ be the capability for a flow $f_{s,d}$ between a source s and a destination d. $C_{f_{s,d}}$ consists of two parts, namely $C_{f_{s,d}} = C^0_{f_{s,d}} || C^1_{f_{s,d}}$. Here, $C^k_{f_{s,d}}$ is defined as:

$$\begin{aligned} C^0_{f_{s,d}} &= \text{Hash}(\text{IP}_s, \text{IP}_d, K^1_R) \\ C^1_{f_{s,d}} &= \text{Hash}(\text{IP}_s, \text{f}(\text{IP}_d), K^2_R) \end{aligned}$$

where IP$_s$ and IP$_d$ are the source and destination IP addresses, K^0_R and K^1_R are the router's secret keys, and f$(\cdot)$ is a function whose output is randomly uniform on [0, n_{max}-1].

$C^0_{f_{s,d}}$ provides identifier authenticity to flows [20, 21], and $C^1_{f_{s,d}}$ restricts the number of per-source capabilities to n_{max} by taking f(IP$_d$) as a hash input. If $C^1_{f_{s,d}}$ is used for estimating flow bandwidth, flows of high-fanout sources would be identified as high-rate flows.

The above attack-flow identification measures help estimate the proportion of legitimate flows in flows carrying S_i, which we define as the *protocol conformance* of S_i and denote by $\mathcal{E}^{\mathcal{P}}_{R_i}$.

Based on the bandwidth and protocol conformances, the *conformance estimate* $\mathcal{E}_{R_i}$ of S_i, representing the estimate of S_i's goodput ratio, is defined as:

$$\mathcal{E}_{R_i} = e^{-\frac{\gamma \cdot \lambda_{S_i} |\mathcal{S}|_{max}}{C_R}} (\mathcal{E}^{\mathcal{B}}_{R_i} - \mathcal{E}^{\mathcal{P}}_{R_i}) + \mathcal{E}^{\mathcal{P}}_{R_i}$$

$$\mathcal{E}_{R_i}(t_j) = (1-\alpha)\mathcal{E}_{R_i} + \alpha \mathcal{E}_{R_i}(t_{j-1})$$

where γ and α are the *weighting coefficients*.

The conformance estimate of S_i is the weighted average of the bandwidth conformance and the protocol conformance, where the weighting factor exponentially favors the protocol conformance as sufficient requests have been made.[8] We determine $\mathcal{E}_{R_i}$ at time t_j by taking the moving average of $\mathcal{E}_{R_i}$s, and update it once in every aggregation period (Δ_{agg}); i.e., $t_j - t_{j-1} = \Delta_{agg}$.

6.2. Aggregation Problem

For path aggregation, the congested router R_0 builds the traffic tree $\mathcal{T}_{R_0}$ using the path identifiers carried in the *active* flows and decomposes it as a legitimate tree $\mathcal{T}^{\mathcal{L}}_{R_0}$ and an attack tree $\mathcal{T}^{\mathcal{A}}_{R_0}$. $\mathcal{T}^{\mathcal{L}}_{R_0}$ is constructed with legitimate path-identifiers that have higher conformance estimates than a certain threshold ($\mathcal{E}_{th}$), and $\mathcal{T}^{\mathcal{A}}_{R_0}$ is constructed with the other (non-legitimate) path-identifers. Then, the router constructs a new traffic tree $\mathcal{T}'_{R_0}$ by merging those two trees at the root (i.e., the disjoint union of $\mathcal{T}^{\mathcal{L}}_{R_0}$ and $\mathcal{T}^{\mathcal{A}}_{R_0}$). Path aggregation is performed on this new traffic tree $\mathcal{T}'_{R_0}$, so that legitimate paths would never be aggregated with attack paths.

The congested router starts path aggregation from neighboring domains (i.e., domains with longest suffix-matching path-identifiers) to localize attack effects, and proceeds with aggregation until a desired number of path reductions are made (viz., Eq. (6.1)). Aggregation is performed with respect to the conformance estimates of paths since link-access guarantees should not be biased by the request rates of paths. Hence, if the number of access-guaranteed path-identifiers is $|\mathcal{S}|_{max}$, the path aggregation problem is to construct an optimal tree which has $|\mathcal{S}|_{max}$ distinct paths and to which providing link-access guarantees maximizes goodput ratio at the congested link. This can be defined as a constrained optimization problem below.

Let $\mathcal{R}$ be the set of all nodes in $\mathcal{T}'_{R_0}$, and $\mathcal{R}_i$ be the set of leaf nodes of a subtree rooted at $R_i \in \mathcal{T}'_{R_0}$ (i.e., $\mathcal{T}_{R_i}$). Then, the optimization problem is defined as:

$$\max O(\mathcal{T}'_{R_0}) = \sum_{R_i \in \mathcal{R}} \frac{1}{|\mathcal{R}_i|} \sum_{R_j \in \mathcal{R}_i} \mathcal{E}_{R_j} \qquad (6.1)$$

$$\text{subject to} \sum_{R_i \in \mathcal{R}} I_{\mathcal{R}_i} \leqslant |\mathcal{S}|_{max} \text{ and } \bigsqcup_{R_i \in \mathcal{R}} \mathcal{R}_i = \mathcal{R}_0$$

where I_{R_i} equals 1, if paths are aggregated at R_i, and 0, otherwise. For a non-aggregated path, I_{R_i} is 1 at the leaf node. Since $\sum_{\mathcal{S}_i \in \mathcal{S}} I_{\mathcal{R}_i}$ is the number of path identifiers seen at R_0, it should be bounded by $|\mathcal{S}|_{max}$.

In the above equation, aggregation at R_i decreases the total conformance estimate by $\frac{|\mathcal{R}_i|-1}{|\mathcal{R}_i|} \sum_{R_j \in \mathcal{R}_i} \mathcal{E}_{R_j}$. We define this value as the *aggregation cost* and denote it by $C^{\mathcal{A}}(R_i)$; i.e., $C^{\mathcal{A}}(R_i) = \frac{|\mathcal{R}_i|-1}{|\mathcal{R}_i|} \sum_{R_j \in \mathcal{R}_i} \mathcal{E}_{R_j}$. Hence, a set of nodes at which aggregating path-identifiers produces the minimum (total) aggregation cost, would be a solution to the above problem.

We note that, if the set of aggregating nodes (routers) are fixed, the optimization problem of Eq. (6.1) is the same as the *0-1 knapsack problem*[9] which is known to be NP-complete. In Eq. (6.1), however, the set of aggregating nodes and the relative aggregation cost of a leaf node ($\frac{|\mathcal{R}_i|-1}{|\mathcal{R}_i|}\mathcal{E}_{R_j}$, $R_j \in \mathcal{R}_i$) vary as aggregation proceeds to the root. This means the 0-1 knapsack problem should be solved repeatedly as the set of aggregating nodes is redefined. We present an efficient algorithm for this problem below.

6.3. Aggregation Algorithm

Whenever aggregation is necessary (i.e., $|\mathcal{S}| > |\mathcal{S}|_{max}$), aggregation is performed as summarized in Algorithm 1. Let $\mathcal{O}$ be the solution set and $\mathcal{C}$ be the candidate set. Initially, $\mathcal{O}$ is empty and $\mathcal{C}$ has all intermediate (i.e., non-leaf)

[8] An insufficient number of requests from a domain could bias the domain's protocol conformance; e.g., unexpected packet drops of a low-rate path-identifier would result in a very low protocol conformance.

[9] $\frac{C^{\mathcal{A}}(R_i)}{|\mathcal{R}_i|}$ can be considered as the unit value of an element, $|\mathcal{R}_i|$ as the size of an element, and $|\mathcal{S}| - |\mathcal{S}|_{max}$ as the knapsack size in the 0-1 knapsack problem.

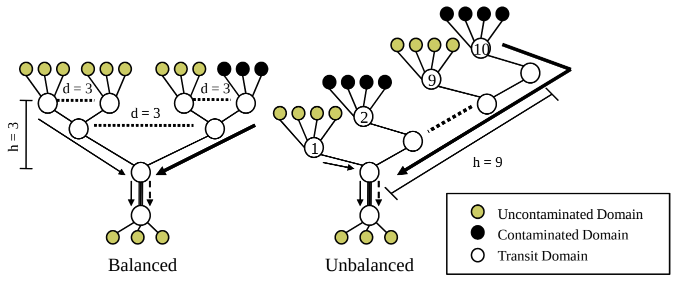

Figure 3: Topology used in simulation.
Legend: "d" is the number of sibling nodes and "h" is the tree height.

nodes in $\mathcal{T}'_{R_0}$ as its elements. Then, the algorithm works in a *greedy* fashion: for each iteration, the node that causes the lowest cost-decrease to $\mathcal{C}$ is added to $\mathcal{O}$, and this continues until the constraint on the number of path identifiers in Eq. (6.1) is satisfied. Though Algorithm 1 is a greedy approximation algorithm, it ensures that the total cost of the candidate set decreases minimally at each iteration. As a consequence, its approximation error from the optimal aggregation cost is bounded by the number of incoming links of the last added node to $\mathcal{O}$. Due to space limitation, the proof of this is not included, but can be found in [9].

Algorithm 1 Aggregation

```
1: Set $\mathcal{O} = \emptyset$ and $\mathcal{C} = \{R_i | R_i \in \mathcal{T}'_{R_0} - \mathcal{R}_0\}$.
2: Move the lowest aggregation cost node in $\mathcal{C}$ to $\mathcal{O}$.
3: $R_i \in \mathcal{C}$ replaces the current solution set if it satisfies the following replacement conditions:
     • $C^{\mathcal{A}}(R_i) < \sum_{R_j \in \mathcal{O}} C^{\mathcal{A}}(R_j)$
     • $C^{\mathcal{A}}(R_i) > \max_{R_j \in \mathcal{O}} C^{\mathcal{A}}(R_j)$
4: Repeat steps 2 and 3 until the constraint on the number of path-identifiers (in Eq. (6.1)) is satisfied.
```

7. Simulation Results

In this section, we present our ns2 simulation results for various attack scenarios to evaluate our design. Network topologies for simulations are configured to capture the worst case effect of different attacks and to ascertain how well our design goals are satisfied. The balanced tree shown in Fig. 3 is used for simulations that evaluate the access guarantees and the effectiveness of aggregation. The unbalanced tree is used to show that our scheme effectively provides access guarantees to domains independently of their location on a routing path. We assign 5% of link capacity to the capability request channel as in [21]. In most simulations, the total request rate of legitimate sources is set close to the link capacity of request channel (i.e., $\rho_{S_i} \approx 1$ for legitimate domains) to accurately capture the effects of attacks. Requests are randomly placed during the specified simulation interval to approximate Poisson arrivals.

We compare our simulation results with those of TVA [21], which protects capability requests using a hierarchical fair-queueing algorithm.

7.1. Link-Access Guarantees

To evaluate the local effect of flooding attacks in our scheme, we use a 27-path balanced tree, where 30 legitimate sources are attached to each leaf node, and attack sources are increased at a leaf node. In this simulation, we set the number of access-guaranteed paths ($|\mathcal{S}|_{max}$) to 27 and the buffer size to that of 108 packets so that 4 buffer-slots are guaranteed to each path. Each source randomly starts 100 different sessions (which is equivalent to 100 times more sources) between 0 and 10 seconds. This source configuration is used for entire simulations. We also run simulations with a TVA [21] router configured to have 1000 queues of length 4 (as TVA requires distinct queues for individual sources in the current implementation) for comparative evaluation.

As Fig. 4 shows, the request drop ratios of legitimate paths are stable over the wide range of attack strengths with both our scheme and TVA. That is, both schemes effectively localize flooding attacks when compared with the *no defense* case. Note that a per-client defense would have the same result as that of no defense when bots are used to flood the link. Yet, our scheme outperforms TVA with a much smaller buffer (108 vs. 4000 buffer-slots). This is because our scheme dynamically adjusts virtual-queue lengths in a *min-max* manner, which in effect allows more than the guaranteed buffer-slots to path-identifiers unless their bursts are synchronized (in which case, only the guaranteed buffer-slots hold).

To illustrate the robustness of the guarantees that our scheme provides, we configure an extreme adversarial scenario where 60 paths of a 64-path balanced tree (i.e., $h = 3$ and $d = 4$ in Fig. 3) send a large number of requests, and observe the service ratio of the remaining 4 paths. Fig. 5 shows the probabilistic guarantee ($\mathcal{G}(|\mathcal{S}|, k, S_i)$, viz., Eq. (5.1)), the stationary service probability ($P(|\mathcal{S}|, k, S_i)$)[10], and the simulation result ($Pr(|\mathcal{S}|, k, \mathcal{S}^{\mathcal{L}})$) for the set of legitimate path-identifiers $\mathcal{S}^{\mathcal{L}}$, under specified bandwidth utilizations – the ratio of request rate to an allocated bandwidth. Even under this extreme attack scenario, the service ratio of legitimate paths is close to the theoretical stationary packet service probability, which is much *higher* than the probabilistic guarantees, as illustrated in the figure.

Next, we show that link-access guarantees provided by our scheme are *independent of attack location*. For this simulation, we use a 40-path unbalanced tree shown in Fig. 3. We attach 30 legitimate sources to each leaf node, and 200

[10]For k guaranteed buffer-slots, the stationary packet service probability of S_i is determined by $P(|\mathcal{S}|, k, S_i) = 1 - \frac{\rho^k_{S_i}(1-\rho_{S_i})}{1-\rho^{k+1}_{S_i}}$. This is derived from the blocking probability of a M/M/1/k queueing system.

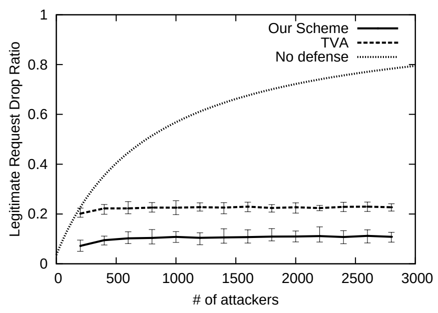

Figure 4: Request drop ratio of legitimate paths. Error bars represent 95% confidence intervals.

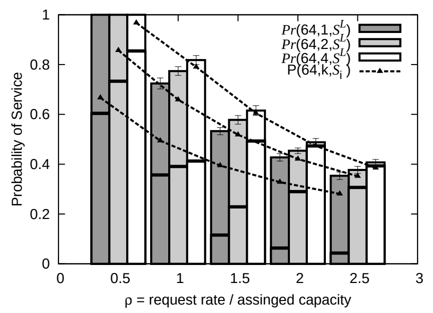

Figure 5: Request service probability of legitimate paths with respect to bandwidth utilization (ρ). The solid horizontal lines inside bars represent the probabilistic guarantees ($\mathcal{G}(|\mathcal{S}|, k, S_i)$).

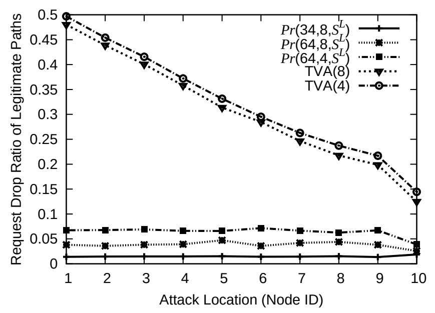

Figure 6: Request drop ratio of legitimate paths with respect to attack location in the unbalanced tree. TVA(k) represents the result of TVA with queue-length k.

attack sources to each of eight attack nodes; four of these nodes are placed at different locations for each simulation and the remaining four nodes are placed at the farthest location from the flooded link. In this scenario, we simulate the queue implementation for $\mathcal{G}(34, 8, S_i)$, $\mathcal{G}(64, 4, S_i)$ and $\mathcal{G}(64, 8, S_i)$, and those for the corresponding 4 and 8-slot queues in a TVA router (i.e., 4000 and 8000 total buffer-slots respectively). Fig. 6 shows the request drop ratios of legitimate paths, where the horizontal axis represents the index of attack location (viz., unbalanced tree in Fig. 3). With our scheme, the request drop ratios are uniform over different attack locations. This means our scheme provides almost same protection against flooding attacks regardless of the attackers' location. In contrast, TVA's performance is highly dependent upon attackers' location since TVA assigns more buffer space to nearby domains (viz., Section 2).

7.2. Aggregation Effect

Path-identifier aggregation, which optimizes domain bandwidth allocation when attack sources are widely dispersed across domains, occurs whenever the number of active paths ($|\mathcal{S}|$) becomes greater than the number of access-guaranteed paths ($|\mathcal{S}|_{max}$). In Fig. 6, the result of the queue implementation for $\mathcal{G}(34, 8, S_i)$ illustrates the effectiveness of aggregation. As aggregation increases bandwidth allocation to legitimate paths by a factor of $\frac{|\mathcal{S}|-|\mathcal{S}|_{max}}{|\mathcal{S}|_{max}}$ (i.e., $6/34 \approx 17.6\%$ in that simulation), the request drop ratio of those paths decreases 76.8% (from 6.43% to 1.49%) when compared with that of the queue implementation for $\mathcal{G}(64, 4, S_i)$ (under which no path aggregation occurs). This is *far below* the stationary drop probability of legitimate paths (i.e., $1 - P(|\mathcal{S}|, 8, S_i) \approx 5.32\%$) which would result when physically separate queues are assigned to those paths.

We also evaluate the effectiveness of the protocol conformance measure in aggregating attack paths. For this, we configure a 64-path balanced tree such that the same number of nodes are attached to leaf nodes to make the request rates of all paths identical. Then, we set $|\mathcal{S}|_{max}$ to 34 (which limits the number of attack path-identifiers by at most two) and increase the fraction of attack sources whose capability requests are denied at the destination host, from 10 to 100% in half of the leaf nodes. Note that the bandwidth conformance measure alone cannot distinguish attack paths from legitimate ones when the same request rates occur in all paths.

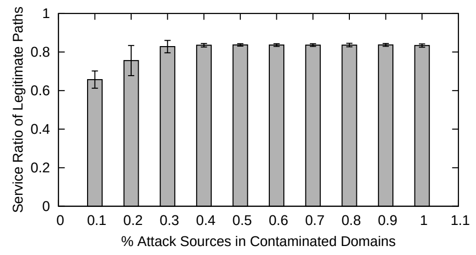

Figure 7: Aggregation by protocol conformance: The request service ratio of legitimate paths increases as the fraction of bots becomes higher.

As Fig. 7 shows, aggregation is more precisely performed on attack paths (which leads to higher service ratios of legitimate paths) as the fraction of attack sources in contaminated domains grows. When domains are lightly contaminated (i.e., the fraction of attack sources is less than 40% in this simulation), legitimate paths can be aggregated. This is because aggregating attack paths near the attack target (i.e., multi-level aggregation of those attack paths) produces a higher aggregation cost than aggregating legitimate paths near their origins. Relatively high cost of multi-level aggregation also causes high service-ratio variation to legitimate paths, as a result of imprecise distinction between legitimate and attack paths.

7.3. Rolling Attacks

Another simulation we performed is that of the "rolling attacks", whereby attack sources change their location to exploit delays in the response time of any defense mechanism. For this simulation, we attach 16 attack nodes at 4 different locations in the unbalanced tree (i.e., at node 1,2,9 and 10) of Fig. 3 and place 200 attack sources in each attack node. We configure a rolling attack such that attack sources attached to node 1 and 10 flood the target for 10 seconds

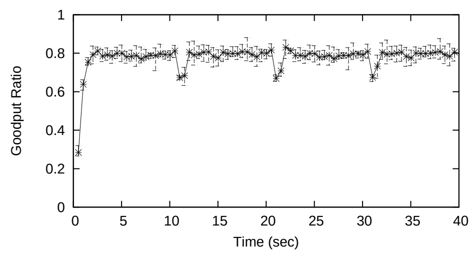

Figure 8: Time variation of goodput ratio at the congested link.
Legend: Error bars represent the minimum and maximum of goodput ratio.

and the other attack sources for the next 10 seconds with a 20-second period.

In Fig. 8, we illustrate the time variation of goodput ratio (viz., Section 6) at the congested link averaged over 10 runs. The goodput ratio is very low at the beginning of the simulation, since attack requests go through the target link before being preempted by legitimate ones. However, as buffer-preemption occurs (as soon as the buffer is filled) and aggregation starts (around $t = 2$), the goodput ratio rises sharply. Changing attack location significantly decreases the goodput ratio as the number of attack path-identifiers at the congested router increases four times (i.e., from 2 aggregated path-identifiers to 8 path-identifiers). However, these effects disappear whenever a new aggregation decision is made on the switched attack paths in Δ_{agg} (which is set to 20·RTT $\approx$ 2 seconds in this simulation).

8. Concluding Remark

In this paper, we present a defense scheme against link flooding attacks targeting connection setups in capability systems. Our design of a new authenticated path-identification mechanism provides individual packets with unforgeable domain identifiers to which link-access guarantees are provided at remote routers. Guarantees of link access, defined as the probabilistic lower bounds of link access, are provided in a domain basis and they are provided differentially based on domain contaminations. Simulation results show the effectiveness of our design: link-access guarantees that are independent of global attack sources and their location, and resilience against attack dispersion via differential guarantees. Differential link-access guarantees would provide positive incentives to administrative domains that employ strong security measures against malware contamination.

Acknowledgment

This research was supported in part by US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001 and by the US Army Research Office under Contract W911NF-07-1-0287 at the University of Maryland. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, US Army Research Office, the U.S. Government, the UK Ministry of Defense, or the UK Government.

References

[1] http://www.computerworld.com/s/article/9076278/.

[2] http://www.potaroo.net/tools/asn32/.

[3] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial-of-service with capabilities. In *Hotnets-II*, 2003.

[4] K. Argyraki and D. R. Cheriton. Network Capabilities: The Good, the Bad and the Ugly. *HotNets IV*, 2005.

[5] D. Dagon, C. Zou, and W. Lee. Modeling Botnet Propagation Using Time Zone. *Network and Distributed System Security Symposium*, 2006.

[6] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *IEEE/ACM Transactions on Networking*, 1998.

[7] W. C. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness. In *INFOCOM*, pages 1520–1529, 2001.

[8] P. Ferguson. Network Ingress Filtering:Defeating Denial of Service Attacks which employ IP Source Address Spoofing. *RFC 2827*, 2000.

[9] S. B. Lee. Localizing the effects of link flooding attacks in the internet. In *Ph.D Thesis, University of Maryland*, 2009.

[10] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *ICNP '01*, 2001.

[11] P. E. McKenney. Stochastic fairness queueing. In *INFOCOM*, 1990.

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

[13] S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson. Practical network support for IP traceback. In *SIGCOMM*, 2000.

[14] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM '95*, pages 231–242, 1995.

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

[16] S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, 2002.

[17] A. Studer and A. Perrig. The coremelt attack. In *ESORICS*, Saint Malo, France, September 2009.

[18] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, 2003.

[19] A. Yaar, A. Perrig, and D. Song. Pi: A Path Identification Mechanism to Defend against DDoS Attacks. In *In IEEE Symposium on Security and Privacy*, 2003.

[20] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *Proceedings of the IEEE Security and Privacy Symposium*, 2004.

[21] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *IEEE/ACM TRANSACTIONS ON NETWORKING*, 2008.