Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters

Christian Esteve Rothenberg
School of Electrical and Computer Engineering
University of Campinas, Brazil
chesteve@dca.fee.unicamp.br

Petri Jokela, Pekka Nikander,
Mikko Sarela, Jukka Ylitalo
Ericsson Research, NomadicLab, Finland
{firstname.lastname}@ericsson.com

Abstract—**In this paper, we propose and analyze an in-packet Bloom-filter-based source-routing architecture resistant to Distributed Denial-of-Service attacks. The approach is based on forwarding identifiers that act simultaneously as path designators, i.e. define which path the packet should take, and as capabilities, i.e. effectively allowing the forwarding nodes along the path to enforce a security policy where only explicitly authorized packets are forwarded. The compact representation is based on a small Bloom filter whose candidate elements (i.e. link names) are dynamically computed at packet forwarding time using a loosely synchronized time-based shared secret and additional in-packet flow information (e.g., invariant packet contents). The capabilities are thus expirable and flow-dependent, but do not require any per-flow network state or memory look-ups, which have been traded-off for additional, though amenable, per-packet computation. Our preliminary security analysis suggests that the self-routing capabilities can be an effective building block towards DDoS-resistant network architectures.**

I. Introduction

An old question is whether routing should happen in a hop-by-hop manner or as source routing. With the proliferation of the Internet, hierarchical hop-by-hop routing won the day. Unfortunately, that came with a price: the network serves more the sender than the receiver. The hop-by-hop approach makes its best to deliver a packet to the destination, whether the receiver wants to receive it or not, opening a venue for unwanted traffic. Various remedies, such as firewalls, deep packet inspection (DPI), and explicit capabilities, have been proposed to address the problem, with variable success.

Network capabilities, as introduced by Anderson et al. [1], are architectural approaches that enable secure statements attached to packets, allowing routers to easily check if a packet has been approved by the receiver. They are typically based on cryptographic approaches that enable routers to verify packets in a stateless way, though some statements, such as those related to a maximum bandwidth, do require state [24]. When capabilities are required, any prospective sender must first retrieve a suitable capability, either directly from the receiver (using explicit bandwidth reserved for that), out of band, or through a trusted third party [19].

In this paper, we present a system where there is no need to have capabilities separate from forwarding identifiers; i.e., where the capabilities work as a forwarding identifier and vice versa. Building upon LIPSIN [10], a native multicast forwarding method based on *in-packet Bloom filters (iBF)*, we introduce a DDoS resistant forwarding service. We construct a system where having separate capabilities is unnecessary, as with our iBF-based forwarding identifiers it becomes computationally hard to extract path information for constructing new capabilities, without insider help.

Addressing Denial-of-Capability (DoC) attacks [2], which aim at preventing new (legitimate) capability-setup packets from reaching the destination by overwhelming the system with capability requests, is out of scope of this work. Recent work has shown effective means to mitigate DoC attacks. For instance, in addition to treating capability request packets as best-effort packets [1], [22], the system can allocate scarce link bandwidth for connection establishment packets based on per-computation fairness (cf. Portcullis puzzle system [14]), or by allowing a well provisioned third party service to act as capability service [19]. Depending on the needs of the service to be contacted, such third party may require cryptographic identity verification, monetary payment or a guarantee, or require a user to solve a hard AI problems e.g. CAPTCHAs [18].

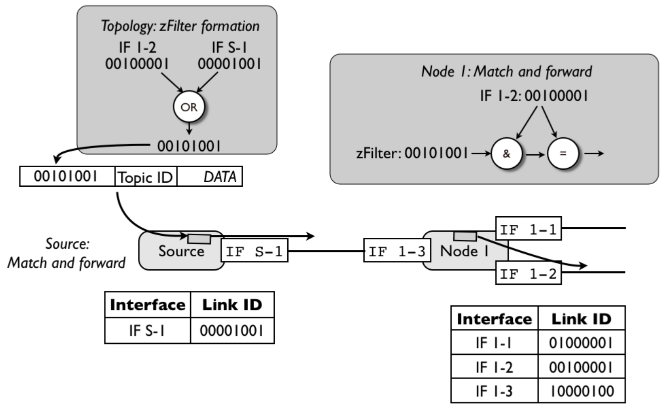
Our approach differs from existing capability-based systems in that (1) the capability has a *fixed size*, independent of the number of hops,[1] (2) the capability acts also as a *forwarding identifier* in a stateless fashion with no forwarding table lookups, and (3) the system is *multicast* friendly.

The rest of this paper is organized as follows. In Sec. 2, we briefly recap the LIPSIN forwarding approach and discuss associated DoS vulnerabilities. In Sec. 3, we describe our proposed DDoS-resistant enhancements. Sec. 4 contains a statistical analysis of the DDoS-resistance properties. In Sec. 5, we briefly discuss related work, concluding the paper in Sec. 6.

II. Background

Using the approach of [16], we divide networking into three components: rendezvous [17], topology [25], and forwarding [10]. The rendezvous is responsible for matching sources and sinks and for instructing the rest of the system. From the security point of view, it acts as a capability distribution center [19], [5]. The division between the topology and the forwarding components is similar to those of the routing as a service proposal [12] and the direct network control approaches, such as 4D [23]. We believe such a division can

[1]There is a practical upper limit (of ≈ 40) for the number of hops; see [10] for a detailed analysis.



Interface	Link ID
IF S-1	00001001

Interface	Link ID
IF 1-1	01000001
IF 1-2	00100001
IF 1-3	10000100

Fig. 1. zFilter creation and forwarding in LIPSIN.

be achieved with a (distributed) topology service, similar to the Path Computation Entity (PCE) [8] in (G)MPLS.

Within this networking model, there are three main avenues for Distributed Denial-of-Service (DDoS) attacks. (1) An attacker may try to attack 'legitimately', i.e. through rendezvous (gaining one or more forwarding identifiers). (2) An attacker may try to overload the rendezvous system with excess requests. (3) The attacker may try to guess or construct a forwarding identifier so that it can overload the target, without receiving help from the rendezvous or topology components. In this paper, we focus on the last one, relying on existing work on capability-based systems, over provisioning, and contractual relationships to solve the former two [1], [22], [14]. Additionally, we exclude insider threats (e.g., Byzantine routers), leaving them for future work.

A. The LIPSIN forwarding mechanism

The LIPSIN [10] forwarding solution does not name nodes or interfaces. Instead, links are named, separately in each direction. Consequently, each forwarding identifier is essentially a set of Link IDs, denoting a delivery tree or a path compressed as a Bloom filter [4] called zFilter.

In practise, each Link ID is an m-bit long string with k bits set to one, with $k \ll m$, and m relatively large. This makes Link IDs statistically unique. For instance, with $m = 256$ and $k = 5$, we get $\approx m!/(m-k)! \approx 10^{12}$ different Link IDs.

The Link IDs are used at two distinct instances. First, to construct a zFilter for a given delivery tree T, the topology component takes a binary OR over the IDs of the links forming the tree (see Fig. 1). The resulting zFilter Z_T is then passed to the source, allowing it to send packets along the delivery tree using the zFilter as the forwarding identifier. Second, when a forwarding node receives a packet, it needs to determine where to forward the packet to. For each outgoing link o, the node checks if the zFilter Z_T contains 1s in those bit positions where the Link ID L_o does. If so, the node forwards the packet along that link; i.e., if $(Z_T \wedge L_o) \equiv L_o$, then forward the packet over o. If the zFilter contains multiple outgoing Link IDs, then the packet is forwarded to each of them, resulting in multicast. Also, as is well known, using Bloom filters introduces the possibility of false positives; their probability rises as more links are included in the iBF.

The Link ID Tag (LIT) mechanism, also described in [10], provides control over the false positives by defining d different names for each outgoing link. Consequently, any given delivery tree can be described with d different iBFs, each of them having different bit patterns. This allows iBF selection based on different criteria, such as fewer false positives.

B. Remaining forwarding vulnerabilities

There appears to be a few vulnerabilities that the basic LIPSIN approach, and any naive source routing forwarding scheme, does not protect from. First, while a given zFilter works only from its source to its sink(s), the same zFilter can be used also for other traffic that what it was meant for. We call this a *zFilter replay attack*. Second, while the used encoding helps to hide the link identifiers, correlation between iBFs is still possible, creating a *computational attack*; see below. Third, while each zFilter is directly usable only by the source and any *en-route* nodes, if an attacker can figure out another zFilter that passes through any of the en-route nodes, it can *inject* traffic to the delivery tree.

In the computational attack, an attacker collects valid, related zFilters and analyses them. Wherever the bit patterns are similar among a group of zFilters, it is likely that any reoccurring bits represent a partial graph common to those zFilters. Hence, knowledge over a large number of $\langle$source, sink(s), zFilter$\rangle$ triples may allow an attacker to create valid zFilters towards a target. By merging correlation pairs from multiple sites (e.g., using bots), DDoS attacks might well be possible. While the introduction of the LIT construction makes this attack computationally more expensive, especially when d is large, the attack appears to remain practical.

To analyse the difference between the original zFilter proposal and our proposal, we introduce a formal security model in Sec. IV. There we evaluate a few attacks in terms of success probabilities and associated costs.

III. Secure in-packet Bloom Filters

To address the above-described security problems (and potentially other, still undiscovered ones), we now propose a system in which the link names are periodically changed and tied to the path and to an upper layer flow identifier. By this we mean an identifier that upper layer, e.g. transport, uses to identify packets belonging to different applications. The link names (and consequently the zFilters) are tied to upper layer flow identifiers to ensure that only packets belonging to a flow requested or approved by some application is delivered. The flow identifier can, but does not need to, be based on IP addresses. Any identifier that upper layers decide to use suffices e.g., topic ID in pub/sub systems.

Instead of relying on a set of pre-defined (but perhaps time-dependent) names for each link, our solution is based on dynamically computing link names depending on the packet contents, the path the packet is taking, and perhaps other context-dependent parameters.

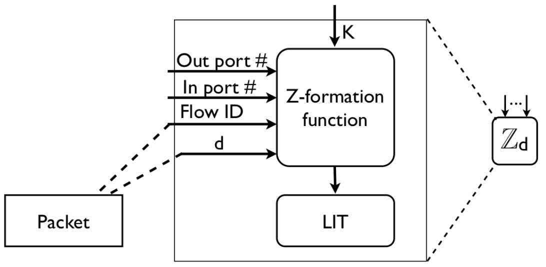


Fig. 2. z-Formation function.

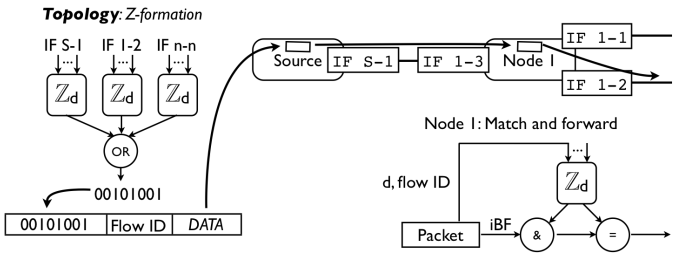


Fig. 3. z-Formation creation and check in the forwarding node.

A. z-Formation

The key idea of the z-Formation is to enable forwarding decisions in a completely dynamic, computational fashion where the iBF, the packet content, and the processing context is used to determine where the packet should be forwarded, if at all. Instead of maintaining a fixed forwarding table containing the Link IDs (or LITs) for each outgoing interface, the *z-Formation* dynamically computes the names of the candidate links on a packet-basis. A function Z computes the LITs using (i) some in-packet information I (a Flow ID), (ii) a periodically changing secret K, (iii) the incoming and outgoing interface indices (In, Out), and (iv) the Link ID Tag index d (see dynamic LIT computation in Fig. 2). As in LIPSIN, also here each LIT $O = Z(I, K(t_i), In, Out, d)$ is a Bloom mask of m bits. As the zFilter is now constructed using these dynamic LITs instead of static LITs, the resulting zFilter becomes additionally bound to the Flow ID, a specific time period, and the input port. Especially, having the Flow ID I as an input parameter ties the given zFilter to only those packets carrying the specified Flow ID, which, for example, makes reactive filtering an easier task (cf. Sec. IV-D).

To construct the time-bound shared secrets K, each forwarding node i shares a master key K_i^m with the topology manager. For any time period t, forwarding nodes compute $K_i(t) = F(K_i^m, t)$, where F is cryptographically secure pseudo-random function. For example, t may be a seeded counter or wall time clock at a coarse enough granularity; in either case, the forwarding nodes and the topology manager need to have loosely synchronized clocks [6]. The topology manager always uses the current valid value of t while forwarding nodes also accept j (one or a few previous) values. In this way, if t is advanced every Δt seconds, even if $K_i(t)$ is compromised for a specific t, the attack is limited to the single forwarding node using the key and to the maximum time of $j\Delta t$.

Finally, as Z takes in both the *outgoing* and *incoming* interface indices as inputs, any given zFilter is tightly bound to the corresponding forwarding path or delivery tree. That is, this feature blocks the injection attack, preventing off-path attackers from sending data towards a delivery tree even if they know both the Flow ID and the zFilter. Additionally, including the incoming interface index as an input-parameter allows us to introduce *virtual interfaces* within forwarding nodes, thereby enabling on-path services.

Secure iBF generation: Identical to the LIPSIN approach, we expect the topology component to generate a zFilter as a result of a path formation request. Using the Flow ID I from the request, the current secret keys $K_i(t)$ and the required interfaces from the nodes in the computed network graph (cf. [8], [12]), the topology component applies Z for each d. Note that as partial results can be easily combined, this operation may be distributed along multiple (e.g., per-domain) topology components. Given the final d candidate iBFs, the topology component picks the best one and hands it over to the source. Section III-B elaborates on how zFilters updates can be pre-computed, requested by receivers, and redistributed to sources every Δt seconds.[2]

Secure iBF forwarding: When a data packet arrives at a forwarding node, the node extracts d and I from the packet. With I, d, the incoming interface index, and the current $K_i(t)$ (plus optional j older) value(s), it computes the LIT for each outgoing link. If the iBF matches the on-the-fly generated LIT, the packet is forwarded along the interface. Dynamic LIT computation can be easily done in parallel for each interface. Note that forwarding nodes are freed from storing any per-flow state or traditional FIB table lookups. Only the *seed* of the secret K and the current accepted values need to be maintained.

Z-function implementation: The Z-function can be implemented as a stream-cipher-like construction, tailored to give constantly out $\approx k$ 1-bits instead of the usual average of $\approx m/2$ 1-bits. Internally, the function may resemble a keystream generator, initialized with a combination of the values K, I, d, In, Out. As typical stream ciphers can be implemented in hardware with only a few shift registers and logic gates (see e.g. [9], [21]), we surmise that the needed circuit could work at full OC-768 line speed.

B. Updating zFilters

If a flow is valid for longer than $(j-1)\Delta t$, the source needs a new zFilter once the old one is about to expire. However, we defer the specific description and evaluation of updating zFilters to future work and merely note that there are at least two methods for doing so. The sink can indicate its willingness to continue receiving by responding to the traffic and instructing the intermediate forwarding nodes to construct an up-to-date zFilter en-route. This can be embedded in upper layer messages, e.g. in transport protocol acknowledgements.

[2]Due to the ability to combine partial results, each part of the network may have a different Δt.

Alternatively, a source can obtain a new zFilter from the rendezvous system. This is similar to other rendezvous-based solutions (e.g., HIP RVS servers [11], mailboxes in [7]). It enables both the sender and the receiver to express their interest in extending their communication. The topology system needs to be instructed to construct the new zFilter, typically using the same path but with the current value for each K_i along the path. As $K_i(t+1)$ can be generated locally, zFilter updates can be easily pre-computed and conveyed to the source (even before the oldest valid key expires), which can use it as the new routing capability for data delivery.

C. Applicability to IP networks

We believe that the z-Formation could be used with IP networks, though there are still many open issues related to NATs and other middle-boxes, partial deployment, and security, to name a few. The 5-tuple $\langle IP_{src}, IP_{dst}, P_{src}, P_{dst}, prot.\rangle$ can serve as the flow ID and the z-Filter will be used for routing decisions within those ASes already deployed. The best place for the z-Filter is likely to be as an extension header after the IPv6 header (or as an IP protocol on top of IPv4).

A group of interconnected ASes can use zFilters to route all traffic within and coming to the group. If a packet without a zFilter comes, it will be sent to the rendezvous system, which will determine whether an approval in the form of zFilter tied to the used 5-tuple will be given to the sender. Support for zFilters can be implemented as a shim layer between IP and transport, or as a separate proxy. Assuming, the victim does not respond to attack traffic, even a single, or a few ASes can reduce the severity of attack by several orders of magnitude as shown in Section IV-B. Additionally, attacks based on copying a single zFilter to multiple attackers are simple to filter, as they require the same 5-tuple in all packets.

IV. Analysis

Preventing an attacker from injecting large numbers of packets without approval via rendezvous is a necessary, though not sufficient condition for DDoS resistant architecture. We now evaluate the effectiveness of the z-Formation forwarding mechanism when malicious nodes try to compromise the network availability by injecting unwanted traffic. First, we introduce the mathematical framework to describe the forwarding model. Then, we use probabilistic methods to quantify the achievable levels of DoS protection. Later on, we discuss the effects our architecture has on replay attacks and computational attacks.

A. Label-based forwarding model

Let $G = (V, E)$ be a network graph, in which all edges are named with Link IDs, i.e. directional bit vectors $e = \{0,1\}^m$, where the number of 1s in each bit vector $e(v_i, v_j) \in E, v_i, v_j \in V$ is exactly k and the 1s are randomly distributed. Let $p(v_0, v_n)$ be a path in the network such that $\langle v_i, v_{i+1}\rangle \in E, \forall i < n$.

Define zFilter $z = \{0,1\}^m$ as an m-bit long string with maximum density ρ_{max} such that zFilter $z_1 \wedge z_2 = z_1 \iff z_1 \subseteq z_2$ and edge $e(v_i, v_j) \in z, \iff e \wedge z = e$. Thus, a

TABLE I
Capacity in terms of edges before reaching the maximum allowed iBF density

ρ_{max}	m = 256		m = 196		m = 128		$fpr_{k=5}$
	#e	#e_{opt}	#e	#e_{opt}	#e	#e_{opt}	
0.45	30	34	23	27	15	17	1.85%
0.50	35	39	27	31	18	20	3.13%
0.55	40	44	31	35	20	22	5.03%

path is encoded in the zFilter if each edge within that path is encoded in the zFilter. We write $p(v, w)$ to denote the smallest zFilter that encodes path p, i.e. $p(v_0, v_n) = \bigcup_{i=1}^{n} e(v_{i-1}, v_i)$.

Assumptions: No forwarding node on any path is hostile. The Link IDs are random and uniformly distributed, that is the secret value K is random and the Z-formation has the property that if K is random, $O = Z(K, I, ...)$ is sufficiently random. The z-Formation produces a different Link ID for each link depending on the incoming interface of the packet. This effectively adds a single hop to the length of the path that the attacker must guess. We use the above described formalism with constant Link IDs in the analysis.

B. Attack description and evaluation

We assume the attacker v_m knows a legitimate zFilter $z(p(s,t))$ between a non-malicious source v_s and target v_t and show that it is difficult for the attacker to create a valid zFilter from malicious nodes to target with it. $V_p = (v_s, ..., v_t)$ is the set of nodes on path p.

Given z with a fill rate of ρ, the number of possible edges e included is $\binom{\rho \cdot m}{k} \approx 3 * 10^8$ for $m = 256, \rho = 0.5$ and $k = 5$. With no way to test off-line for the validity of single edges within z, the first attack strategy consists of randomly trying a set of $z_i \supseteq z$ to see if it can deliver packets through, i.e. if $p(v_m, v) \in z_i, v \in V_p$. Hence, the attack model is based on *brute force* and is equivalent to a randomly generated z causing false positives along the path(s) toward the set of nodes V_p (as the path V_p is included in each zFilter z_i). However, if a known z is used at its maximum capacity ρ_{max}, the attacker cannot set additional bits to include extra edges in z.

In both cases, we can assume that z has $\rho_{max} * m$ bits set to 1. The threat consists of $p(v_m, v), v \in V_p$ being encoded in z. The probability of a non-included edge having its k bits set to one in any z depends only on the fill factor and is equal to ρ_{max}^k. Table I shows the estimated false positive rate and the average number of edge labels, with and without the $d = 8$ LIT optimization, that can be inserted before reaching ρ_{max}.

When an attacking node is h hops away from any node in V_p, the attacker needs to cause h false positives to get the packet forwarded to and through the valid path. Thus, the probability of a successful attack, i.e. $z \wedge p(v_m, v) = p(v_m, v), v \in V_p$, is equal to $Pa = \rho_{max}^{k \cdot h}$. Figure 4 shows, at the left axis, the estimates of Pa for different maximum fill factors and attack path lengths h. An attack path length of one means that the attacker is a neighbor of some node on path. In this case, the probability of falsely forwarding a forged packet

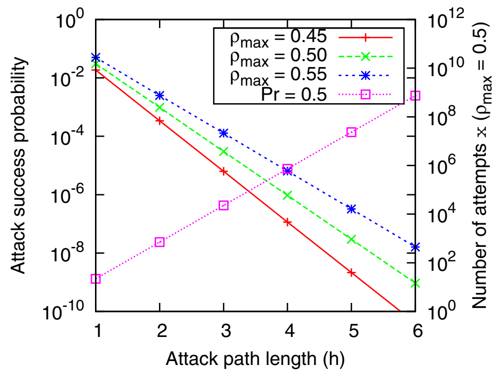


Fig. 4. On the left axis, attack success probability for different ρ_{max}. On the right axis, the line with square points represents the attempts required to success with probability 1/2.

is around 3%. When the malicious node v_m is more hops away from the valid path, this probability sinks drastically, i.e. 10^{-9} for v_m to guess a working label path over 5 hops.

Finally, we can determine how expensive it is to guess a zFilter from v_m to any other target node v_t. We estimate how many attack attempts (x), consisting of randomly generating maximally filled z, are required to have some probability Pr of obtaining at least one valid path label to a destination v_t h hops away from v_m (the right axis on Fig. 4 shows $Pr = 0.5$):

$$x = \log_{1-Pa}(1 - Pr) \quad (1)$$

C. Discussion

Tying the zFilter to a set of periodically changing keys, one for *each* forwarding node, makes replay attacks less severe, as they can only be used during the joint lifetime of the keys. It also makes computational attacks more difficult. Even if the attackers know the full topology of the network, tying the zFilter to each forwarding node's secret key reduces the best attack strategy to a brute force attack consisting of generating random labels and hoping that at least one of them reaches the target(s). At the same time, the seed-based re-keying scheme is local and introduces low communication overhead (low frequency seed exchanges) between forwarding nodes and distributed topology instances, which can easily anticipate to zFilters update requests.

Assuming that the attacking node is capable of injecting 10^6 packets per second (e.g., 1Gbps edge link and 1000 bits per packet), a malicious node will need over 40 minutes to guess, with probability 1/2, a working label for a 5-hop path. If the receiver of the traffic does not answer to such packets, then the system reduces the magnitude of attack traffic by the percentage of packets filtered enroute. An attacking host sending packets 2 hops away from the target, with $\rho_{max} = 0.5$, would only be able to get (approximately) 0.1% of the attack traffic into the path. Thus, changing keys as slow as once every 20 minutes, the forwarding plane can be protected for paths longer than 4 hops with very high probability.

Under some circumstances, we may want to shorten the time for which any host can receive unwanted traffic, empowering the receiver by means of the capabilities renewal mechanisms. For instance, a re-keying frequency of around 1 minute would be short enough to (i) protect very short paths and to (ii) limit the duration of DDoS attacks based on the misuse of legitimate zFilters. Typically, an expiration interval in the order of a few dozens of seconds is long enough to complete average transactional traffic without requiring zFilter renewal.

Note that we have not only assumed a very high and constant attack traffic injection rate (10^6 pps) but also the existence of a return channel to know whether randomly generated z reach the intended victim(s). As a final remark, the DDoS protected forwarding plane only complements additional security measures at the end node higher level stacks similar to end-host firewall implementations where only solicited (subscribed) data flows are allowed and processed.

D. Attack detection and mitigation

By virtue of the time-based re-keying mechanism, a forged path lasts only for $j\Delta t$ in the worst case. After that, a malicious node would need to re-initiate the attack process. As the most efficient attacks we are aware of require excessive probing, an attack can be detected early by the sudden increase on false positives caused by the falsely labeled packets injected by the attacking node(s). Hence, a blacklist mechanisms can be used to block or shape down any suspicious traffic. By definition of an in-packet flow identifier (I), each attack needs to be tailored for a specific I. This does not only limit the scope of an attack but also eases any blacklisting mechanisms based on I.[3]

V. Related Work

Anderson et al. [1] were the first to propose in-packet capabilities. More generally, [1], [5], [7], [24] are all close to our approach in the sense that for sending the sender needs a permission from the receiver. Compared to our work, the main difference is that in our case the forwarding identifier (bound to an upper layer flow identifier), acts by itself as the capability; additional capability fields or cryptographic end-to-end schemes are not needed.

SANE [5] was, to our knowledge, the first proposal to combine centralized computed source routes and capabilities together. It achieves that, by encrypting each hop in layers, with a big routing identifier of $10 + 14 \cdot hops$ bytes; e.g. a 13-hop path would require 1536 bits. Therefore it cannot be considered scalable to Internet-wide scales.

Our secure fast-path forwarding mechanism is close to the stateless path pinning service provided by SNAPP [13], where IP forwarding decisions are cached into the flow into the flow initiating packets. The chain of securely constructed forwarding directives is returned to the sender who is now able to use them as packet headers enabling fast switching decisions and additional benefits from the separation of routing from forwarding, including sender-controlled paths, expensive

[3]Of course, if the underlying link technology offers any trustful node identification mechanism, the filter rule can be built on the network node identity of the malicious node.

route lookups, sender anonymity, and accountability. A key difference of our solution is using an in-packet Bloom filter to encode the pinned path, whith the main performance benefits of a fixed header size independent from the number of hops at the cost of some amount of false positives. While our presented networking model involves a rendezvous service using topology information, an en-route capability formation similar in spirit to IP switching could be easily supported.

Also advocating for the separation of routing and forwarding, Platypus [15] proposes a capability-based system to enable authenticated and policy-compliant IP source routing. Functionally similar to our Z-function is their usage of a distributed temporal secret to verify flow binding capabilities at line speed. However, our capabilities are not transferable per design as they securely embed the routing information.

Ballani et al. [3] were the first to use in-network Bloom filters for pro-actively filtering distributed denial-of-service attacks. Our forwarding plane attains a similar off-by-default behaviour but does this by matching the communication interest of sources and sinks over a slow-path rendezvous network instead of propagating over the network routing infrastructure explicitly reachability directives signaled by end hosts.

Phalanx [7] combines capabilities with a multi-path-aware overlay and shows that Bloom filters can be used to reduce state requirements while providing probabilistic guarantees for in-network security. Yang et al.propose TVA [24] , a capability approach that shows how the router state can be bounded even when routers enforce bandwidth limits for capabilities.

In [20], Wolf presented a mechanism where packet forwarding is dependent on credentials. If the packet does not contain a correct credential, it will not be forwarded. The credentials are packed into a small in-packet Bloom filter and each router verifies that its credentials are included. Compared to ours, in their work routing is based on traditional IP addresses and the credentials are issued for a specific IP 5-tuple flow.

VI. Conclusion

In this paper, we have proposed a source-routing-based packet forwarding mechanism that is highly resistant to distributed denial-of-service attacks. In essence, in our approach the hosts have no names; only links are named. To be able to send, any prospective sender needs to acquire a forwarding identifier that simultaneously acts as a path capability. By delegating capability creation to a rendezvous component that explicitly considers both the senders' and receivers' interest, we shift power from senders to receivers. In our approach, receivers are in control of what they want to receive.

To our knowledge, our approach is the first that combines forwarding identifiers and capabilities in an efficient way, thereby creating *self-routing capabilities*. By virtue of the Bloom-filter-based construction of the capabilities, even the links names remain statistically undisclosed. By binding the routing capabilities to specific flows, time periods, and network paths, we create a high barrier for attackers, making it hard to forge valid capabilities.

While the forwarding operation in our scheme is computationally heavier than in LIPSIN, we surmise that it still can be implemented in line-speed hardware, as no memory lookups are required, the number of logic gates appears to be moderate, and the required dynamic per-packet state can be stored in simple shift registers. The verification of this assumption remains as future work.

VII. Acknowledgements

Writing of this paper was supported by TEKES (Finnish Funding Agency for Technology and Innovation) as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

References

[1] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. *Hotnets II*, pages 39–44, 2004.

[2] K. Argyraki and D. Cheriton. Network capabilities: The good, the bad and the ugly. *ACM HotNets-IV*, Jan 2005.

[3] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default. *ACM HotNets IV*, Aug 2005.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.

[5] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: a protection architecture for enterprise networks. In *USENIX-SS'06*, Berkeley, CA, USA, 2006.

[6] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, 1981.

[7] C. Dixon and T. Anderson. Phalanx: Withstanding multimillion-node botnets. *Usenix NSDI*, 2008.

[8] A. Farrel, J.-P. Vasseur, and J. Ash. A path computation element (PCE)-based architecture. IETF RFC 4655, 2006.

[9] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj. Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. SASC'08, Feb 2008.

[10] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander. LIPSIN: Line speed publish/subscribe inter-networking. In *Proceedings of ACM SIGCOMM'09, Barcelona, Spain*, Aug. 2009.

[11] J. Laganier and L. Eggert. Host identity protocol (HIP) rendezvous extension. Technical report, RFC 5204, April 2008.

[12] K. Lakshminarayanan, I. Stoica, and S. Shenker. Routing as a service, 2004.

[13] B. Parno, A. Perrig, and D. Andersen. Snapp: stateless network-authenticated path pinning. In *ACM ASIACCS '08*, 2008.

[14] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *Sigcomm*, 2007.

[15] B. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. *SIGCOMM CCR*, 34(4):167–178, 2004.

[16] M. Särelä, T. Rinta-aho, and S. Tarkoma. RTFM: Publish/subscribe internetworking architecture. ICT Mobile Summit, 2008.

[17] S. Tarkoma, D. Trossen, and M. Särelä. Black boxed rendezvous based networking. In *MobiArch '08*, 2008.

[18] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. *LNICS*, pages 294–311, 2003.

[19] D. Wendlandt, D. Andersen, and A. Perrig. Fastpass: Providing first-packet delivery. *Technical report CMU cylab*, 2006.

[20] T. Wolf. A credential-based data path architecture for assurable global networking. In *IEEE MILCOM*, 2007.

[21] T. Wollinger, J. Guajardo, and C. Paar. Security on FPGAs: State-of-the-art implementations and attacks. *ACM Trans. Embed. Comput. Syst.*, 3(3):534–574, 2004.

[22] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate DDoS flooding attacks. *Security and Privacy*, 2004.

[23] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai. Tesseract: A 4D network control plane. In *NSDI'07*, 2007.

[24] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-limiting network architecture. *IEEE/ACM Trans. on Networking*, 16(6):1267–1280, 2008.

[25] A. Zahemszky, A. Csaszar, P. Nikander, and C. Esteve. Exploring the pubsub routing/forwarding space. In *ICC FutNet09*, 2009.