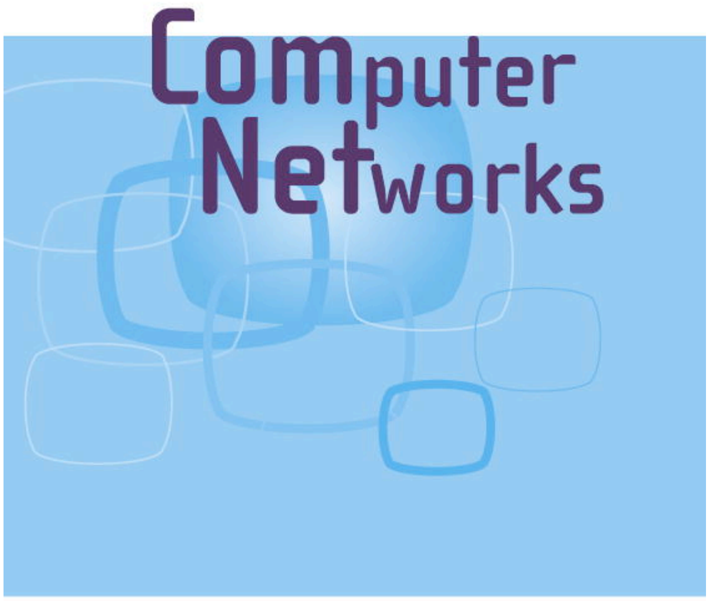
Volume 52 Issue 2 8 February 2008 ISSN 1389-1286



Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 52 (2008) 399–417
Computer Networks
www.elsevier.com/locate/comnet

Learning the valid incoming direction of IP packets [☆]

Jun Li [a,*], Jelena Mirkovic [b], Toby Ehrenkranz [a], Mengqiu Wang [c], Peter Reiher [c], Lixia Zhang [c]

[a] *Department of Computer and Information Science, University of Oregon, Eugene, OR 97403, United States*
[b] *Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292, United States*
[c] *Computer Science Department, UCLA, Los Angeles, CA 90095, United States*

Received 16 March 2007; received in revised form 14 September 2007; accepted 26 September 2007
Available online 9 October 2007

Responsible Editor: R. Molva

Abstract

Packet forwarding on the Internet is solely based on the destination address of packets, and it is easy to forge the source address of IP packets without affecting the delivery of the packets. To solve this problem, one can have routers check whether or not every packet comes from a correct direction based on its source address field. However, due to routing asymmetry in today's Internet, a router cannot simply reverse its forwarding table to determine the correct incoming direction of a packet.

In this paper, we present the source address validity enforcement protocol, SAVE, which allows routers to learn valid incoming directions for any given source address. SAVE is independent from—and can work with—any specific routing protocol. By only interfacing with the forwarding table at routers, SAVE allows routers to properly propagate valid source address information from source address spaces to all destinations, and allows each router *en route* to build and maintain an *incoming tree* to associate each source address prefix with a corresponding incoming interface. The incoming tree is further valuable in handling routing changes: although a routing change at one router could affect the incoming direction of source address spaces from many locations, only the router that sees the change needs to send out new updates. Finally, SAVE has a good performance with low overhead.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Incoming direction; IP spoofing; Source address validity

1. Introduction

It is well known that the source address of IP packets in today's Internet can be easily forged. When an Internet router sends traffic towards its destination, the router will forward it solely based on its destination address, no matter what the source address of the packet is. This ease of source address spoofing has helped attackers to hide

☆ This material is partially based upon work supported by the National Science Foundation under Grant No. ANI-9980501. A preliminary version of this article appeared in [1].

* Corresponding author. Tel.: +1 541 346 4424.
E-mail addresses: lijun@cs.uoregon.edu (J. Li), mirkovic@isi.edu (J. Mirkovic), tehrenkr@cs.uoregon.edu (T. Ehrenkranz), mengqiuw@yahoo.com (M. Wang), reiher@cs.ucla.edu (P. Reiher), lixia@cs.ucla.edu (L. Zhang).

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.09.024

themselves and has made innocent end-hosts be both blamed and attacked. Typical examples include DDoS attacks [2], TCP SYN flooding attacks [3], smurf attacks [4], and reflector attacks [5]. Moreover, this problem impairs many source-address-based functions often performed by routers at the core or edge of the Internet; for example, routers may need to perform per-source fair queuing, congestion control, or source-address-based traffic management schemes.

Researchers have studied both reactive and proactive approaches to handling packets with forged IP source addresses. Unfortunately, there are defects in existing approaches. In employing a reactive approach, one can try to trace back the real origin of packets [6–9], or discover that a packet is from a path already identified as an attack path [10,11]. However, packet tracing takes place after an attack is detected, not when it is occurring.

If one already can determine that a packet carries an invalid source address, one could *proactively* filter such a packet right away. For example, if one can assume that routing is symmetric (i.e., the route from a node A to B is the same as that from B to A), one could use a router's forwarding table to determine whether or not a packet with a specific source address arrives from the same interface as forwarding a packet toward that address [12]. However, we know that paths through the Internet are frequently asymmetric [13], which means that the forwarding tables used by routers to deliver packets are not reliable for determining where packets should come from. Or, before a packet leaves a stub network to enter the Internet, one can check whether or not the packet indeed carries a source address from the stub network, i.e., ingress filtering [14]; similarly, for packets reaching a stub network from the Internet, one can apply egress filtering to ensure they do *not* carry a source address from the stub network [15]. But, with the ingress or egress filtering, edge routers can only check a packet's incoming direction with a very coarse granularity (i.e., whether toward or from a stub network), and routers not at the edge cannot help at all. One could also filter packets by checking whether or not a packet carries a specific key or some kind of cryptographic authentication information [16,17], or even a TTL value within an expected range [18,19]. But relying on keys remaining secret is not always safe, and cryptographic operations become too expensive on a per-packet basis; filtering based on invalid TTL values is frequently imprecise.

What is much needed is a reliable, lightweight, and proactive approach that would allow a router to easily verify source addresses of IP packets. Routers currently have forwarding tables that specify the outgoing interface for each destination address space (or destination address prefix). If they also had tables specifying proper incoming directions for source address spaces, then an attacker's choice of forgeable IP source addresses will be sharply reduced. All improperly addressed packets could be easily dropped as soon as the forgeries were detected, and attack-tracing tools can much more easily determine the possible sources of attacks [20]. Moreover, this table could be used for non-security related purposes, such as source-based traffic engineering, congestion control, or fair queuing. A router performing reverse path forwarding (RPF [21]) can track down hop by hop those routers that are on the delivery path from a source to itself, and multicasting protocols (such as DVMRP [22], CBT [23], and PIM [24]) that use RPF to build *reverse* shortest-path multicasting trees now could build *true* shortest-path trees.

In this paper, we present our source address validity enforcement (SAVE) protocol. SAVE builds incoming tables at routers, enabling them to verify whether each received packet has arrived from the expected incoming interface according to the packet's source address. In addition, based on the "incoming tree" concept we will describe, SAVE can help a router learn the path that packets from a specific source address would travel to reach this router. The SAVE protocol aims to have the following properties:

- SAVE is independent of the underlying routing protocol, so that it can easily run on top of different routing infrastructures;
- SAVE should respond to routing changes and adjust incoming table entries at every SAVE router in a timely manner;
- SAVE must be lightweight in order to minimize router overhead and scale well while achieving its goals;
- SAVE needs to cleanly handle various advanced routing techniques in use throughout the Internet (such as mobile IP, tunneling, multipath routing, as well as the inter-domain and intra-domain routing infrastructure);
- SAVE can only be deployed incrementally, and should offer benefits with incremental deployment; and

- SAVE must be secured or attackers could bypass any security it offers or even directly use SAVE to launch certain attacks.

The rest of this paper is organized as follows: Section 2 discusses related work; Section 3 describes the basic approach of SAVE; Sections 4–6 describe the protocol in detail; Section 7 discusses how SAVE deals with advanced routing techniques; Section 8 presents simulation results on the costs of running the protocol and demonstrations of its efficacy; Section 9 touches on remaining issues; and Section 10 concludes the paper.

2. Related work

Source address validity enforcement can be either router-based or end-host-based. We briefly touch upon end-host-based solutions first, then focus on router-based solutions since SAVE is router-based. We also describe hybrid solutions that need participation from both routers and end-hosts, or that both routers and end-hosts can separately apply.

2.1. End-host-based approaches

A variety of end-host-based detection approaches can be found in [19]. These approaches can be further classified as active or passive, depending on if end-hosts actively probe to determine source address validity, or if they simply observe the incoming packets.

However, routers and end-hosts both need the source address of IP packets to be valid (see Section 1). An end-host-based approach will not help routers in this regard.

In fact, although end-host detection is easier to deploy, it is the lack of source address inspection at routers that allows IP spoofing to run wild. Purely end-host-based approaches can indeed help verify the validity of IP source address, but only a router-based solution can prevent packets with an invalid source address from crossing the network and reaching victims.

2.2. Router-based approaches

Below we focus mostly on router-based methods, as SAVE is also router-based. Existing works include forwarding-table-based filtering, ingress/egress filtering, route-based distributed filtering, key-based approaches, and general filtering. Recall the strength of SAVE is that it allows every router to learn and rely on the incoming direction of IP packets in order to validate their source address.

Forwarding-table-based-filtering [12,25] assumes that the outgoing interface that a router uses to reach a given address, as specified by its forwarding table, is also the valid incoming interface for packets originating from that address. Unfortunately, routing asymmetry on the Internet is common [13], invalidating this assumption.

At the border of a stub network, ingress [14] and egress [15] filtering respectively ensure that packets from the stub network and toward the stub network have valid source IP addresses. However, this approach does not provide *every* router with incoming direction knowledge; and for routers at the edge of a stub network, it only can tell whether a packet with a specific source address should be inbound or outbound. SAVE allows every router to learn the correct incoming interface of source addresses.

Route-based distributed packet filtering (DPF) research in [20] studied benefits of DPF filtering for attack prevention and traceback, and partial deployment strategies. Unfortunately, the work assumed the existence of a DPF system without actually designing an approach for routers to learn the correct route for every source address. More recently, researchers have proposed DPF-like implementations, including IDPF [26] and BASE [27]. However, besides being very BGP-specific, IDPF only learns *feasible* paths, not *actual* paths. BASE is also tightly tied to BGP, and it cannot handle AS-level routing asymmetry well.

SPM [17] utilizes keys associated with source-destination AS pairs. Unlike SAVE, it cannot be used for router-based services that require incoming direction knowledge. Designed only for spoofing prevention, SPM is specific to BGP and benefits participating ASes at the AS level. Perhaps most distressing, if attackers learn the correct key for a source-destination pair, they can successfully send spoofed packets from *anywhere* in the network without raising suspicion.

Baker [12] proposes a general filtering approach where many fields, including but not limited to the source address, can be used for filtering. For instance, martian address filtering allows routers to discard packets if their source addresses are special addresses (loopback address, broadcast address, etc.) or are not unicast addresses.

2.3. Hybrid approaches

Besides methods which only routers can implement, or that only end-hosts can implement, other approaches exist which either routers or end-hosts could possibly utilize. Still other approaches require both routers and end-hosts to work together. We describe these approaches below.

One such approach is to apply cryptographic operations to guarantee authenticity of packet information, represented by IPsec [28]. IPsec can be run in two modes, tunnel mode and transport mode. In tunnel mode, packets are protected between two routers, or an end-host and a router. In transport mode, packets are protected between end-hosts. Unfortunately, the high computational cost of cryptographic operations prevent such approaches from being widely employed per packet.

Hop-count filtering [18] relies on spoofed packets traveling over a different number of hops than legitimate traffic. The concept could be applied by either end-hosts or routers. But, with only a small range of possible hop-counts, it has limited efficacy.

Packet tracing has been widely studied [6,29,9], and often involves packet marking [7,8,30–34]. Generally, packet tracing involves both routers and end-hosts: routers must mark packets while end-hosts often decide which packets to trace. While complementary to each other, a fundamental difference between SAVE and tracing is that tracing is typically performed after an attack is detected, possibly too late to avoid damage! Tracing IP packets with forged source addresses requires complex and often expensive techniques to observe the traffic at routers and reconstruct a packet's real path. Many tracing methods become ineffective when the volume of attack traffic is small or the attack is distributed [35].

Similar in concept to packet tracing, routers running Pi [10] or StackPi [11] mark each packet to identify the path that it traveled. End-hosts use the path identifiers to filter out packets which traveled along an identified attack path. Detecting attack paths, and dropping attack packets is up to the end-hosts. Routers cannot filter attack packets, nor can they discover incoming direction information.

3. SAVE primer

3.1. Basic approach

The goal of the SAVE protocol is to allow routers to filter packets with forged source addresses, or reliably perform various source-based functionalities. SAVE accomplishes this by building an *incoming table* at each router that specifies the valid incoming interface for packets with a specific source address.

The information needed to construct an incoming table is inherently different from that used to build a forwarding table. In a routing protocol, routing updates advertise the set of destination address spaces that routers can reach and the properties of the routes used. Each router then uses these updates and some local policies to calculate its best outgoing interface for each destination address space. On the other hand, the SAVE protocol should be designed to inform routers about the path that has already been chosen, in order to allow routers on the path to a destination to deduce valid incoming interfaces for specific source addresses.

In SAVE, every router is associated with a set of source addresses. For packets from these addresses, they must go through this router to reach certain destinations. A router that forwards packets on behalf of hosts in a local area network (LAN) has a source address space that covers addresses of those LAN hosts; a border router of an autonomous system (AS) handles the source address space of the entire AS; and a transit router with no attached hosts has a source address space that consists of all its own IP addresses.

The basic approach of SAVE works as follows. For every entry in its forwarding table, a SAVE router periodically generates SAVE updates toward the corresponding destination address space that travel along the same path as legitimate packets from this router's source address space. Forwarding table changes will also trigger new SAVE updates. In both cases, an update will specify the originating source address space and carries the destination address space. Since every SAVE update arrives on the same incoming interface as the valid IP packets from the source address space of the update, every router *en route* between the source and final destination can record the incoming interface of a SAVE update as the legitimate incoming interface for the source address space of the update. In addition, in order to reduce bandwidth overhead, SAVE allows intermediate routers to piggyback their own source address spaces on a passing-by SAVE update.

3.2. Fundamental challenges

Although the basic SAVE operations are simple, there are several fundamental challenges. Here we discuss two of them: (1) ensuring SAVE updates follow the same path as valid data packets and (2) reacting to routing changes. We cover additional issues related to reducing overhead and handling advanced routing techniques in Sections 6 and 7, respectively.

The first issue is to ensure that the SAVE updates follow the proper packet delivery paths. The key here is that a SAVE update is forwarded toward a destination address space, not a single IP address. The SAVE protocol must account for all paths toward the addresses in a destination address space. In Fig. 1, for the SAVE update that router *B* initiated on behalf of the source address space S_B, if router *A* *only* forwards the update toward router *R*, router *r* will not be able to learn the valid incoming interface for S_B. Instead, in order to ensure *all* downstream routers can learn the proper incoming direction information, the SAVE protocol needs to generate one SAVE update toward router *R*, and one toward router *r*.

The second issue concerns routing changes. Routing changes establish new paths from sources to destinations, and SAVE must make sure the incoming interface information will be up-to-date at every router for every source address space. In the basic approach, SAVE routers that notice a forwarding table change will initiate new SAVE updates. However, not all routers that should generate SAVE updates will necessarily experience a change in their forwarding table. In Fig. 2, router *D* initially chooses router *B* as the next hop to reach address space S_A. The incoming table of router *A* is shown in Table 1a. Assume that due to the failure of link *BD*, router *D* updates its forwarding table so that router *C* becomes its new next hop to S_A. Although *D* will send a new SAVE update to S_A, which indicates to *A* that packets from S_D should now arrive from interface 2 instead, routers *E* and *F* do not change their forwarding entries for S_A, and thus they will not regenerate SAVE updates! As a result, router *A* will have stale information about address spaces S_E and S_F (Table 1b).

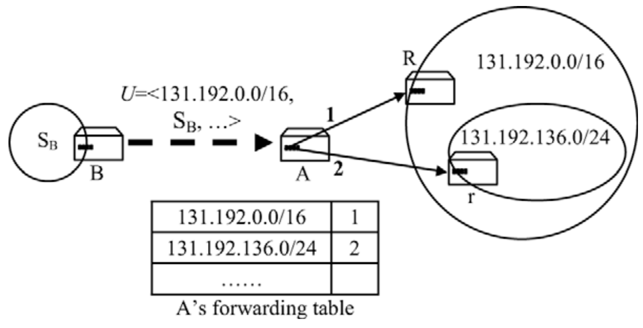


Fig. 1. An example of SAVE update forwarding.

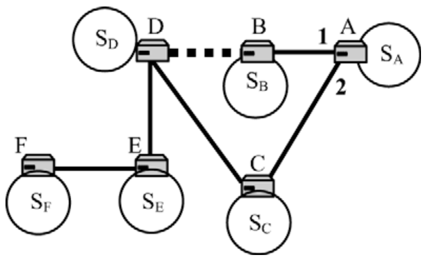


Fig. 2. An example topology of routers and their source address spaces. (S_X stands for router *X*'s source address space.) After link *BD* fails, router *D* changes its route to S_A. A SAVE update is thus triggered at *D* and sent toward S_A, causing router *A* to update its incoming table. But *E* and *F* do not detect the routing change, leaving two stale entries about S_E and S_F in *A*'s incoming table (Table 1).

Table 1
Example incoming tables

Source address space	Valid incoming interfaces
(a) Router A's incoming table before router D's routing change (Fig. 2)	
S_B	1
S_C	2
S_D	1
S_E	1
S_F	1
(b) Router A's incoming table after router D's routing change (Fig. 2)	
S_B	1
S_C	2
S_D	2
S_E	1 (should be 2)
S_F	1 (should be 2)

Periodically sending SAVE updates solves the problem eventually, but not in a sufficiently timely manner. What is needed is that when router *A* receives a SAVE update that changes the incoming interface for address space S_D, router *A* can decide right away that the same change should also be applied to address spaces S_E and S_F. SAVE handles this issue through the use of an incoming tree mechanism, which we describe in Section 5.2.

3.3. Architecture of SAVE

SAVE effectively addresses the above fundamental challenges. While only interfacing with the forwarding table at routers, SAVE allows routers to

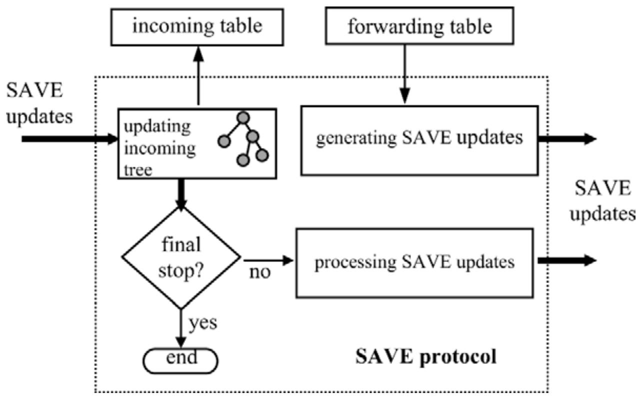


Fig. 3. The architecture of the SAVE protocol.

properly propagate SAVE updates, ensuring SAVE updates follow the correct path; furthermore, SAVE introduces an incoming-tree-based mechanism to learn and store the valid incoming direction information for different source addresses, as well as correctly handle routing changes in a timely manner.

Fig. 3 shows the architecture of the SAVE protocol. SAVE's main components include generating SAVE updates, processing SAVE updates, and updating the incoming tree and incoming table based on SAVE updates. We describe these components in detail below. Section 4 describes the method SAVE uses to communicate incoming direction information. Then Section 5 goes into how a router stores incoming direction information in a novel manner.

4. Communicating incoming direction information

SAVE routers communicate incoming direction information through SAVE updates. In this section, we describe SAVE updates themselves in detail, followed by how the updates are generated and propagated.

4.1. SAVE update

Every SAVE update has a `destination address space` field that specifies the final destination address space of this SAVE update. This field helps downstream SAVE routers correctly propagate a SAVE update. If a SAVE update originated from router R_1 has crossed SAVE routers R_2, $R_3,\ldots,R_n$ in sequence in reaching the destination address space of the update, packets from the source address space of R_1 will also cross R_2, $R_3,\ldots,R_n$ in sequence. (We will describe the SAVE update propagation procedure in Section 4.3.) These crossed routers can then record the incoming interface of the update as the valid incoming interface for packets from the source address space of R_1.

In fact, while heading toward its destination address space, every SAVE update also has an `address space vector` (`ASV`) field to record a chain of source address spaces that are associated with the routers that the SAVE update has crossed in sequence, starting with the origin router of the update. If a SAVE update originated from R_1 has crossed SAVE routers R_2, $R_3,\ldots,R_n$ in sequence, the ASV field will have the form $\langle S_1, S_2,\ldots,S_n\rangle$, where S_i is the source address space of R_i ($i = 1, 2,\ldots,n$). Every router that receives this SAVE update can be certain that valid packets en route from address space S_i, towards the update's destination, will cross R_{i+1}, $R_{i+2},\ldots,R_n$ before reaching itself. (There may be some unlisted routers between R_n and the current router, as we can see in Section 6.1.) In other words, every router along the path of a SAVE update can record the incoming interface of the SAVE update as the legitimate incoming interface for packets from *every* source address space in the ASV of the update.

Finally, to reduce bandwidth overhead by SAVE updates, in certain circumstances a SAVE router can append its update on a passing update. In other words, every SAVE update also has an `appendable` flag to indicate whether downstream SAVE routers can append their source address spaces to this update's ASV field. We will illustrate this flag in detail in Section 6.1.

4.2. Generating SAVE updates

A SAVE router generates SAVE updates for every entry in its forwarding table. If router R has source address space S_R and has a forwarding entry for destination address space D, the corresponding SAVE update will be: $\langle$**destination address space** $= D$, **ASV** $= \langle S_R\rangle$, **appendable** $=$ **true**$\rangle$. At this point, the ASV field only contains the source address space of R, but the next SAVE router on the path to D will be allowed to append its source address space since the appendable flag is true.

This SAVE update will head toward D. In particular, it will be forwarded along the outgoing interface specified in the forwarding entry for D. Along with the update propagation process that will be illustrated in Section 4.3, this is necessary to ensure that the update will take the same path as packets from this router's source address space.

SAVE supports both triggered updates and periodic updates. Not only will a SAVE router periodically initiate SAVE updates, but whenever it detects a change in a forwarding table entry, the router will also trigger a SAVE update corresponding to this entry. The pseudocode in Fig. 4 describes steps for generating SAVE updates.

4.3. *Propagating SAVE updates*

Unless a SAVE router is the last hop for every address from the destination address space of a received SAVE update, the router is an intermediate hop for this update and needs to further propagate it downstream. In order for the SAVE update to reach its destination address space, the router will forward the update in the same way as it would forward regular packets toward the destination address space of the update. Doing so, the incoming interface of the SAVE update at every downstream SAVE router will be the same as—thus also recorded as—the incoming interface of all source address spaces carried in the ASV field of the update.

An intermediate SAVE router uses its forwarding table to decide how to forward a SAVE update. Denoting D the destination address space of a SAVE update, recall that the update should travel along the same path as regular packets toward D. However, the router may not have a forwarding table entry that points exactly to D. The router could have one or more forwarding table entries that point to the sub-spaces of D, or a forwarding entry that points to a super-space of D. To forward a *packet* with a specific destination address in D, the router needs to choose the most specific forwarding table entry that matches the destination address of the packet. Thus, in order for an *update* to travel along the same direction as packets towards all addresses of D, the update must be forwarded according to all such forwarding table entries.

As described in the pseudocode in Fig. 5, SAVE is designed to propagate a SAVE update as follows:

- For every forwarding entry that specifies a route toward a sub-space of D, the router will create a new SAVE update, which is a duplicate of the original SAVE update except that the destination address space in the new update will be set to this sub-space. The router then needs to recursively propagate the new SAVE update in the same way as the update in question. Here, creating new updates will result in splitting the original SAVE update into multiple SAVE updates.
- If there are no sub-space forwarding entries, or all sub-space forwarding entries together do not cover the whole space of D, the forwarding entry toward the smallest super-space that covers D—which could be the entry that exactly points to D if it exists—will be used to forward the SAVE update. This is because this forwarding entry will be used for forwarding data packets toward D, or toward the part of D not covered by sub-space forwarding entries. Note that in this case the destination address space of the SAVE update does not change.

Finally, if a passing update is appendable according to its appendable flag (we describe the flag in Section 6.1), an intermediate SAVE router will append its own source address space to the update's ASV field before further forwarding it downstream. The ASV field allows downstream SAVE routers to record and adjust the incoming interface of packets from not only the source address spaces in the ASV field, but also other source address spaces. We illustrate this concept in Section 5.

```
Procedure generateUpdates(): SAVE update generation at router R.
    $S_R$ : router R's source address space

loop:
for each forwarding entry $e$:
    <destination prefix, outgoing interface $oif$>
    if (should_generate_SAVE_update_for($e$))
        compose SAVE update $U$:
            $U \leftarrow$ <destination prefix, ASV=<$S_R$>, appendable=true>
            send $U$ out along interface $oif$
```

Fig. 4. SAVE update generation pseudocode.

```
Procedure propagateUpdate($U$): SAVE update $U$ propagation at router R.
    $U$ = <$S_D$, ASV, $appendable$>, where ASV=<$S_1, S_2, \cdots, S_k$> ($k \geq 1$)
    $S_R$ : R's source address space

if (R is the last hop to reach every address in $S_D$)
    return

if ($S_R \supseteq (S_1 \cup S_2 \cup \cdots \cup S_k)$ ) /* U is a replaceable SAVE update */
    return

Define set $E$={forwarding entry $e_i$ = <$S_{Di}, oif_i$> |
    $S_{Di} \subset S_D$ && $\neg\exists e_j$=<$S_{Dj}, oif_j$> that $S_{Di} \subset S_{Dj} \subset S_D$ }
        /* first-level sub-space forwarding entries */
for every $e_i$ in $E$
    create a SAVE update:
        $U_i$ ←<$S_{Di}$, ASV, $appendable$>
    propagateUpdate($U_i$) /* process and propagate $U_i$ */
end loop

Define set $S \leftarrow \cup S_{Di}$, for all <$S_{Di}, \cdots$>$\in E$
if ($S == S_D$)
    return /*Great! The entire $S_D$ is covered using $E$*/

/* find smallest super-space forwarding entry covering $S_D$*/
find $f$: <$S_{D'}, oif'$> that $S_{D'} \supseteq S_D$ && $\neg\exists e_j$=<$S_{Dj}, oif_j$>: $S_{D'} \supset S_{Dj} \supseteq S_D$
if ($f$ is not found)
    return

if ($appendable$) {
    ASV←<ASV, $S_R$> /* append $S_R$; ASV=<$S_1, S_2, \cdots, S_k, S_R$> */
    if (R has processed $f$, i.e. already generated SAVE update for $f$)
        appendable←false
}

forward $U$=<$S_D$, ASV, $appendable$> along outgoing interface $oif'$
```

Fig. 5. Recursive SAVE update propagation procedure.

5. Storing incoming direction information

SAVE routers employ two data structures for keeping track of incoming direction information: the incoming table and the incoming tree. The incoming table at a SAVE router is simple: it contains entries that specify valid incoming interfaces for different source address prefixes. The incoming tree, which is in fact used to create the incoming table, is slightly more complicated.

5.1. Incoming table

The incoming table needs to be fast and efficient since it must be inside a router's fast path. Its simple concept and similarity to forwarding tables helps in this regard; routers are already highly optimized for table lookups for forwarding tables, and this optimization can also help when using incoming tables to look up valid incoming direction information.

The incoming table is further optimized to reduce the storage cost by leveraging symmetries in network routing. When routing is symmetric, i.e., a router uses the same interface for both forwarding packets *to* an address space and receiving packets *from* that space, the router's forwarding table entry for that space can also serve as the incoming table entry for that space. In other words, the router does not need to create a new incoming table entry for that space. Otherwise, when routing is asymmetric, the router adds a flag to the forwarding table entry to indicate that the incoming table must be consulted to determine the correct incoming interface. The degree to which this optimization saves storage space depends on the degree of routing asymmetry present.

5.2. Incoming tree

The incoming tree at a SAVE router serves two purposes: (1) to derive the incoming table of that router, and (2) to maintain the relationship of different source address spaces. The former is straightforward: Every node on the tree represents a source address space and is mapped to the valid incoming interface for that address space (except the root of the tree). Note a source address space can include one or more address prefixes. For the latter, the incoming tree is designed to have the following properties:

- If—and only if—a SAVE update crosses router A and then router B before reaching a router R, node S_A will be the child of node S_B on R's incoming tree. Here S_A and S_B are A's and B's source address space, respectively. Note that B and R could be separated by other routers, or are the same router. For defining parent/child relationships, S_R is treated as the root node of router R's incoming tree.
- As a natural result of the first property, the incoming interface that a child node is mapped to is determined by—and the same as—the incoming interface that its parent is mapped to.
- Recursively applying the second property, all nodes from a sub-tree that is directly below the root are mapped to the same incoming interface, making building an incoming table from an incoming tree even more straightforward.

Consider router A in Fig. 2. Its incoming tree before link BD fails is shown in Fig. 6a, where S_D is the parent of S_E, and S_E is the parent of S_F. Triggered by the routing change at D, S_D's new SAVE update will cross C before reaching A, causing A to modify its incoming tree so that S_D becomes the child of S_C, and all source address spaces of D, E, and F will now correctly map to interface 2 (Fig. 6b).

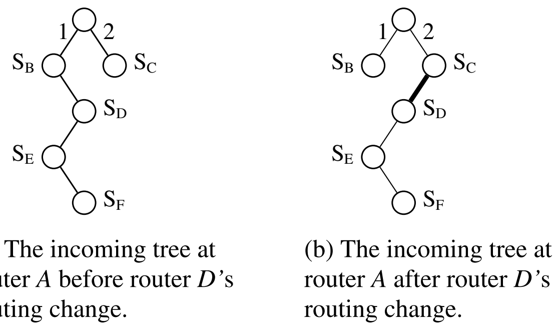

1
2
SB
SC
SD
SE
SF
(a) The incoming tree at router A before router D's routing change.
1
2
SB
SC
SD
SE
SF
(b) The incoming tree at router A after router D's routing change.

Fig. 6. Incoming tree example for topology in Fig. 2.

Another important property of an incoming tree is that not only can it help determine the "incoming interface" for packets from a specific source, but also the "incoming path". For example, Fig. 6b indicates that packets from S_E will pass through router D, and then C, before reaching router A. We leave the possible usage of incoming path knowledge as a topic for future investigation.

5.3. Updating an incoming tree

Upon receipt of a SAVE update, a SAVE router uses the ASV of the SAVE update to maintain its incoming tree, and thus its incoming table (see Fig. 7 for a pseudocode description of the algorithm).

In general, an ASV has the form $\langle S_1, S_2, \ldots, S_n \rangle$, where S_i is the source address space of a SAVE router R_i ($i = 1, \ldots, n$). To preserve the properties of the tree discussed in Section 5.2, the incoming tree updating procedure must first ensure that the ASV will be "grafted" into the tree as an intact branch, where S_i will be the direct child of S_{i+1}. Second, the procedure must map this branch to the incoming interface that the SAVE update arrived on. Third, descendants of S_i on the tree must map to the same incoming interface as S_i.

The tree update procedure therefore parses the ASV in reverse order (see Fig. 7), processing S_n, the last ASV element, first. If S_n is not yet in the tree, it is grafted directly under the root; otherwise, if S_n's existing interface in the tree is not this update's incoming interface, the sub-tree under S_n (not just S_n itself) will be remapped to the new interface and grafted under the root. For any other element of the ASV, S_i ($i \neq n$), given that node S_{i+1} has just been positioned into the tree correctly, the whole S_i sub-tree can be relocated directly under node S_{i+1}. This could map the S_i sub-tree to a new interface.

6. Optimizing SAVE

6.1. The appendable flag

Instead of initiating a new update toward a destination address space S_D, a SAVE router can piggyback the update on a passing SAVE update U that also heads toward S_D. But, if the router has just initiated its own SAVE update toward S_D, it can mark U as non-appendable by setting U's *appendable* flag as false. This router still appends its own

```
Procedure updateIncomingTree($U$): Incoming tree update procedure at router R
    $S_R$: the address space associated with router R
    $U$: a newly received SAVE update
        $U = < S_D$, ASV, $appendable>$,
        where ASV=$<S_1, S_2, \cdots, S_n>$
    $iif$: the incoming interface that $U$ arrives on
    subtree(X): a sub-tree of the incoming tree that is rooted at X

[Initialization when router R boots up]
    The tree only contains the root node

/* handle $S_n$ first */
if ($S_n$ does not exist in the incoming tree)
    graft $S_n$ under the root
    associate $S_n$ with $iif$
else
    if ($iif \neq$ the current interface associated with $S_n$)
        graft subtree($S_n$) under the root
        remap $S_n$ to $iif$

/* now handle $S_{n-1}, S_{n-2}, \cdots, S_2, S_1$ one by one */
for ($i \leftarrow n-1$; $i > 0$; $i$-- )
    if ($S_i$ does not exist in the incoming tree)
        graft $S_i$ under $S_{i+1}$
    else
        graft subtree($S_i$) directly under $S_{i+1}$ (if not yet)
end
```

Fig. 7. Incoming tree update procedure upon receipt of a SAVE update.

source address space, but all downstream routers will stop appending their source address spaces into U's ASV field, thus keeping the size of U from unnecessarily growing.

Now that downstream routers do not always append their source address spaces, the ASV field of a SAVE update does not always map to the entire path that the update has traversed. However, every downstream router will still be able to derive a *complete* ASV that corresponds to *all* the routers that a SAVE update has crossed, as illustrated below.

Assume a downstream router R receives a SAVE update $U(R_1)$ that originated from R_1. The ASV field of $U(R_1)$ is expressed as $\langle S_1, S_2, \ldots, S_n \rangle$, where S_i is the source address space of a SAVE router R_i, and $U(R_1)$ is marked as non-appendable. The non-appendable flag suggests that R_n is an intermediate router and must have already initiated a SAVE update, denoted as $U(R_n)$, toward the same destination address space as $U(R_1)$. R should receive both $U(R_1)$ and $U(R_n)$. Assume the ASV field of $U(R_n)$ is $\langle S_n, S_{n+1}, \ldots, S_{n+m} \rangle$. If $U(R_n)$ is appendable, its ASV will map to the entire path from R_n to R. R can then concatenate the ASV of $U(R_1)$ and the ASV of $U(R_n)$ to obtain a complete ASV, i.e. $\langle S_1, S_2, \ldots, S_n, S_{n+1}, \ldots, S_{n+m} \rangle$. And this ASV maps to all routers on the path from R_1 to R. If R_n's update is marked as non-appendable, R can still successfully obtain the complete ASV for $U(R_1)$ by executing a recursive procedure: learning the complete ASV corresponding to $U(R_n)$. Note that the above concatenation does not happen literally; instead, it is implicit because of the incoming tree update procedure.

6.2. Replaceable updates

As another optimization, SAVE does not need to forward those updates that are *replaceable*. An update is replaceable from the point of view of a specific SAVE router if every address space element in the update's ASV is contained by this router's source address space. This router already has produced or will produce the necessary SAVE updates to carry the information in replaceable updates. This optimization matches well with the two-level routing infrastructure of the Internet: since all packets from an AS to the outside must cross a border router, and the whole AS space is the source address space of that border router, those SAVE updates from within an AS are all replaceable and will not leak beyond the AS. Section 7.2 further discusses AS-level replaceable updates.

6.3. Routing-protocol specific optimization

As described thus far, the design of SAVE is routing-protocol independent. This important aspect allows SAVE to run on top of any routing infrastructure. However, further optimizations are possible if SAVE is allowed to use protocol-specific information.

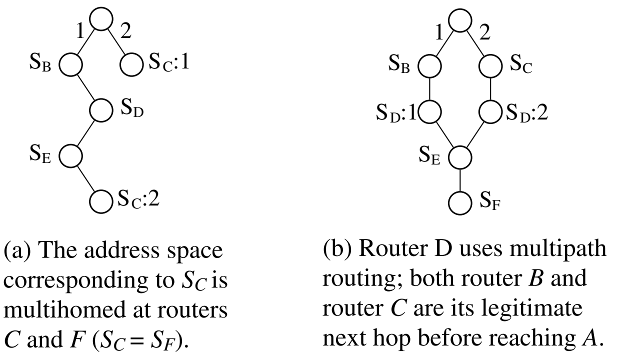
If SAVE only runs on BGP routers, SAVE's overhead can be further reduced. Instead of using network prefixes to represent its source address spaces, a router can simply use its AS number. This optimization could decrease both the storage overhead and the traffic overhead. With a single AS number generally replacing multiple prefixes, the space required to represent a source address space decreases—possibly dramatically if the AS number is replacing many prefixes. Thus, anywhere source address spaces are used, we will see a reduction in SAVE's overhead. Specifically, a router's incoming tree will require less space, and the ASV of a SAVE update will be smaller. (In order to keep a router's fast path as streamlined as possible, AS numbers can be translated into prefixes when building the incoming table from the incoming tree.)

7. SAVE and advanced routing techniques

7.1. Multihoming and multipath routing

With the SAVE protocol implemented as described thus far, problems will arise regarding the use of multihoming and multipath routing. When multihoming or multipath routing is in use, a downstream router may receive multiple SAVE updates for the same source address space, but from different incoming directions. Without any modifications, the incoming tables of downstream routers would only consider packets matching the incoming direction for the *last* SAVE update received to be valid.

To solve this problem, we introduce a simple, optional tag to the source address space entries within the ASV (address space vector) field of SAVE updates. This tag identifies which home or path the update originated from or passed through, respectively. The tag only needs to be unique amongst all of the multiple homes or multiple paths. This is simple to achieve, since when an address space is multihomed, the routers in charge of that space will know; similarly, when a router has multiple outgoing paths for the same destination space, it knows.



1
2
S_B
S_C:1
S_D
S_E
S_C:2
(a) The address space corresponding to S_C is multihomed at routers C and F ($S_C = S_F$).
1
2
S_B
S_C
S_D:1
S_D:2
S_E
S_F
(b) Router D uses multipath routing; both router B and router C are its legitimate next hop before reaching A.

Fig. 8. Incoming tree examples with multihoming and multipath. The topology from Fig. 2 is used.

So, when a router is in charge of a multihomed space, or a router uses multipath routing, it adds the tag for the corresponding home or path to its source address space in any outgoing update's ASV. As described earlier in Section 5.2, each unique source address space in the ASV (now further differentiated by tags) corresponds to a unique node in downstream routers' incoming trees. This allows downstream routers to correctly maintain their incoming trees, and therefore incoming tables.

Fig. 8a and Table 2a show a multihoming example. Similarly, Fig. 8b and Table 2b contain a multipath example. Note, this solution may create incoming trees that are no longer true trees; nodes may have multiple parents. However, since the graph is directional, the functionality is not impaired.

7.2. Hierarchical routing

The concept of replaceable updates (Section 6.2) can be extended to work with the routing hierarchy present in the Internet. Both the intra/inter-AS

Table 2
Incoming tables with multihoming and multipath routing

Source address space	Valid incoming interfaces
(a) The incoming table corresponding to Fig. 8a	
S_B	1
$S_C = S_F$	1,2
S_D	1
S_E	1
(b) The incoming table corresponding to Fig. 8b	
S_B	1
S_C	1
S_D	1,2
S_E	1,2
S_F	1,2

hierarchy for BGP routing and the intra/inter-area hierarchy for OSPF [36] routing can help reduce the size of routing tables, and the intra/inter-AS hierarchy for BGP further allows organizations to hide their internal routing information from outside parties. SAVE can also take advantage of this hierarchy to have smaller incoming trees and incoming tables, while still not exposing internal routing information to outside parties.

The general idea is simple and is as follows. Utilizing the intra/inter-AS hierarchy, all border routers of a given AS act as one "virtual router" with the entire AS as its source address space. Any update from an internal router towards an external destination space will simply be dropped by the border router. If the internal routing change which triggered the internal update also caused an AS-level routing change, the border router will generate an update itself, representing the entire AS. If there is no AS-level routing change, the border router has no reason to generate or forward the internal update towards external ASes; as far as border routers from other ASes are concerned, no incoming directions have changed. Similarly, an update traveling from an external router towards internal destinations is also dropped. As far as an internal router is concerned, the "virtual" router is *always* the incoming direction of all external traffic.

The above optimizations do not change significantly for OSPF intra/inter-area operation. For our purposes, OSPF area border routers function essentially like AS border routers above. All of the area border routers for a single area function as a single virtual router. An area border router does not forward intra-area updates to external areas, nor does it forward inter-area updates to internal routers.

When deployed at the AS-level, further optimizations from Section 6.3 can also be applied. Since all AS border routers use BGP, instead of representing the source address space of an AS using network prefixes, a border router can use the AS number of the domain. In this way, internal routers use network prefixes, while border routers use AS numbers.

7.3. Mobile IP and tunneling

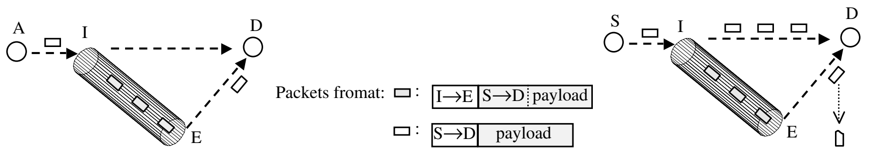
Some Internet traffic does not use default routing behavior, and SAVE must handle such traffic properly. Such cases include mobile IP, tunneling, source routing, etc.

Mobile IP [37] potentially conflicts with SAVE in that a mobile host's packets, if carrying its home IP address, would be rejected whenever the mobile host is outside its home network (since generally it uses a different path to the destination than the rest of its home network). The reverse tunneling technique [38], proposed to handle such conflicts for general address filtering, also works for SAVE. A mobile host's packets are first tunneled from a foreign network back to its home agent, and then forwarded to the destination; thus, the source addresses of those packets are valid on each segment of the path. IPv6 requires that a packet from a mobile host in a foreign network use a care-of address (an address belonging to the foreign network) as the packet's source address [39], thus also solving the problem.

IP tunneling complicates source address validation. A packet's true source address is buried inside a wrapping IP header that contains the source address of the ingress of a tunnel, thus the true internal source address can bypass the validation. Source validation must be performed before a packet enters a tunnel as well as after the packet departs from the tunnel. For example, in Fig. 9a spoofing packets from attacker A enter a tunnel at router I and depart from the tunnel at router E before reaching the destination router at D. If their source address can be found to be spoofing by SAVE routers along the path from A to I or from E to D, these packets can then be caught.

When a SAVE update reaches the ingress point of a tunnel, the update will be encapsulated and then forwarded through the tunnel as a regular data packet. There is no SAVE-related action taken on this update until it is decapsulated at the egress point of the tunnel.

In the view of SAVE, there are two different types of tunnels: those that merely add one level of encapsulation (and perhaps also IPsec for a secure tunnel), which follow the same route as regular data packets, and those that deviate from the regular routing path. The latter type can cause legitimate packets to be dropped. As shown in Fig. 9b, whereas packets from S toward D normally go through router I along path ID to reach D, if I reroutes these packets through a tunnel before reaching D, they will appear to D as arriving from an illegitimate incoming direction. We handle this case using our solution for multipath routing (Section 7.1). When the ingress point of a tunnel (router I in Fig. 9b) recognizes that the tunnel may introduce two legitimate paths toward a destination, it will



(a) Spoong packets from an attacker *A* going through a tunnel.

(b) Legitimate packets going through a tunnel.

Fig. 9. SAVE and IP tunneling.

add a tag to its source address space when forwarding SAVE updates through the tunnel, and another tag when forwarding SAVE updates through a regular path.

8. Performance evaluation

8.1. Simulation design

We have implemented and tested the SAVE protocol in a custom simulation environment. We performed extensive simulation experiments to obtain information related to: (1) whether all spoofed packets can be successfully detected and dropped, (2) whether valid packets are dropped erroneously, (3) the transient behavior of SAVE, and (4) the cost of SAVE.

In the simulation, all routers run the SAVE protocol in addition to routing protocols. Corresponding to the two-level routing infrastructure of the Internet, we used the transit-stub topology generator from GT-ITM [40] to generate inter-domain connectivity and intra-domain connectivity. We simulated both inter-domain routing and intra-domain routing. We also introduced asymmetric routing.

For inter-domain routing we used BGP [41], the *de facto* inter-domain routing protocol. For intra-domain routing, we chose RIP [42] as it is the easiest to implement in simulation. In fact, SAVE is independent of routing protocols; any routing protocol would suffice as long as it correctly creates forwarding tables.

8.2. Effectiveness against spoofed packets

Spoofed packets may escape in the following stringent or rare situations: (1) if a router's incoming table specifies that a range of IP addresses should come in from a particular direction, the router has no way of knowing if a packet with a source address in that range carries a forged source address from the same range; (2) also, prior to the adjustment based on newly received SAVE updates, certain entries at the incoming table of a SAVE router may be temporarily obsolete so that spoofed packets from a wrong direction might be regarded as valid and thus escape.

We performed extensive simulation experiments to verify the correctness of SAVE, focusing on false negatives. Each simulated packet source generates both valid packets and spoofed packets that are controlled by two independent Poisson processes. Spoofed source addresses were randomly chosen from a pool of all source addresses in the network. Every router is under an average load condition and every packet carries a reachable destination address; thus a packet can only be dropped due to address filtering or a transient routing inconsistency caused by topology changes. If SAVE can drop all spoofed packets, the distribution of dropped packets over time should match the generation model of spoofed packets. This was verified in numerous scenarios over different topologies, with the presence of asymmetric routing and dynamic routing changes. We report one such scenario in Fig. 10.

8.3. Correctly identifying legitimate packets

When a forwarding table changes and a new route to a destination address space is being set up, there is a transient period in which the incoming tables are incorrect, due to the delay of generating, forwarding, and processing the triggered SAVE update. During this time SAVE must adjust every incoming table along the new route. If a data packet is sent toward the destination during this period, it could be erroneously dropped even though it carries a valid source address. More accurately, assuming that the propagation delay of a SAVE update is the same as that of a valid data packet, the data packet can be dropped by mistake if:

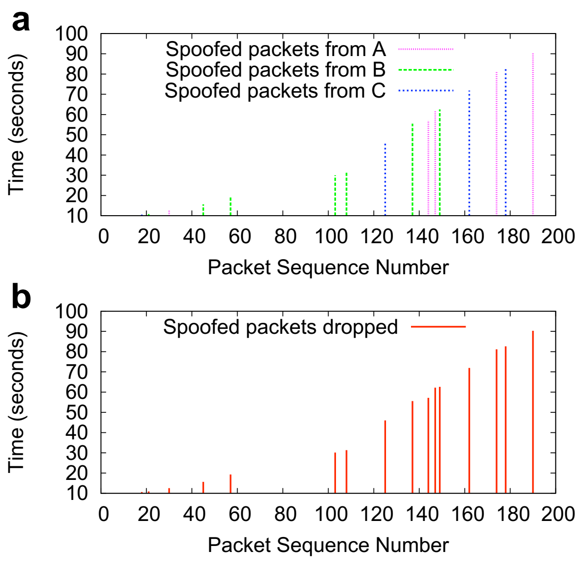


Fig. 10. SAVE effectiveness verification. In this scenario, a DDoS attack is performed from three different machines. Every packet in the simulation has a global unique sequence number. (a) Shows the pattern of spoofed packets generated at three hosts (legitimate packets are not shown). (b) Shows the patterns of packets that are dropped by SAVE. We can observe that all spoofed packets are dropped.

1. The data packet is sent while the SAVE update is still being generated due to a forwarding table change; in this case, the packet can reach downstream routers earlier than the SAVE update, and will be validated using the obsolete incoming information there.
2. The data packet is received at an intermediate router or its final hop while the incoming tree and the incoming table are still being updated using the triggered SAVE update; due to the obsolete entry in the incoming table, the packet will be regarded as a spoofed packet.

Given that both windows above involve only processing delay and are fairly short, we expect that few legitimate packets will be dropped due to stale incoming table entries. In our experiments we experienced no filtering drops of valid packets due to routing changes.

8.4. Storage overhead of SAVE

The incoming table built by SAVE is on the fast path of a router and it is important that the incoming table does not take too much space. We report two different costs for the incoming table: one is the cost when all routes are asymmetric; the other is the cost with an average level of asymmetry resulted from running routing protocols in the simulation. The former case is rare but provides a worst case scenario; in general, the amount of routing asymmetry observed in the Internet is less.

In addition, we also compared the size of the corresponding fast-path data structures at a router: the incoming table used by SAVE and the forwarding table used by routing protocols. This comparison is useful in the real world. As network operators already know the storage cost of routing protocols, they can use these comparisons to further understand the storage of SAVE.

Figs. 11 and 12 show the incoming table size of SAVE as well as its comparison with forwarding tables built by routing protocols. Fig. 11 compares SAVE with RIP for different single-domain topologies. Fig. 12 compares SAVE with BGP for different multiple-domain topologies. Clearly, SAVE incurs

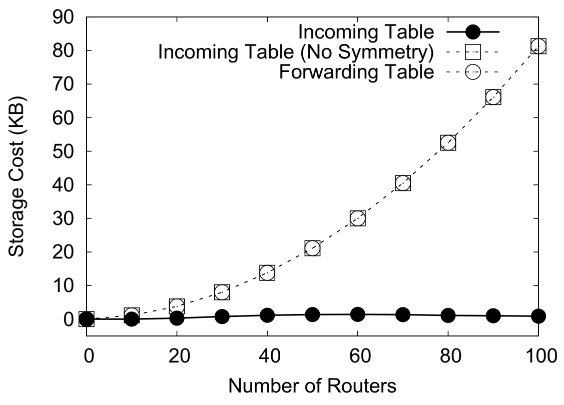


Fig. 11. Storage cost comparison for single-domain topologies.

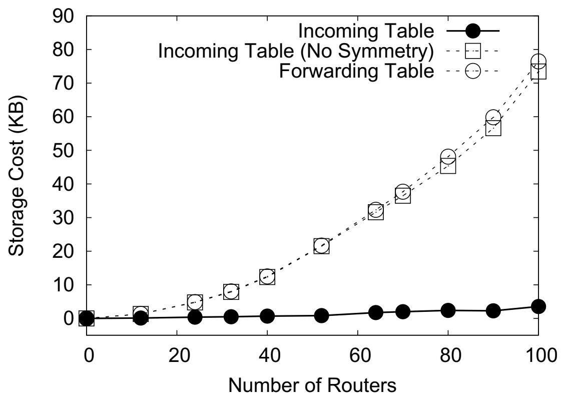


Fig. 12. Storage cost comparison for multiple-domain topologies.

equal or less storage overhead. Figs. 11 and 12 show that, in the worst case, the incoming table is similar in size to the forwarding table; and with more reasonable amounts of asymmetry present in our simulation, the storage cost of the incoming table is very small.

8.5. Control traffic overhead of SAVE

To assess the bandwidth requirements of SAVE, we compared the triggered and periodic bandwidth costs of SAVE and routing protocols. Again, results are presented as a comparison, since it is more meaningful than raw numbers.

Assuming SAVE updates and routing updates are initiated with the same frequency (we use every 30 s), we compared SAVE and RIP in terms of periodic bandwidth cost over single-domain topologies. Simulations over different topologies show similar results (Fig. 13), where 10 different topologies were measured for each given number of routers. The ratio of SAVE bandwidth cost versus RIP bandwidth cost is lower than 1 as the number of nodes in topologies grows beyond 40, suggesting that SAVE has better scaling properties than RIP. We also measured the per-link bandwidth cost of SAVE, which varies with topology. Over those single-domain topologies in Fig. 11, the maximum per-link bandwidth cost varies from 3.2 to 6.9 kbytes/s.

We also compared SAVE bandwidth in multiple-domain topologies with BGP and RIP combined. Because BGP does not initiate periodic routing updates, we compared the bandwidth without periodic transmission of SAVE updates and RIP updates. The result is shown in Fig. 14. In the worst case measured, SAVE uses less than 60% of the bandwidth of BGP and RIP combined. The maximum per-link bandwidth cost here varies from 0.6 to 6.4 kbytes/s over the topologies we used in Fig. 12.

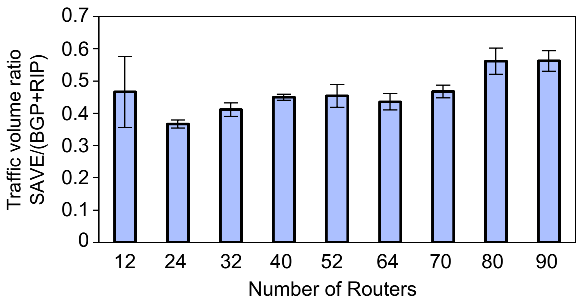


Fig. 14. Bandwidth comparison between SAVE and routing protocols (BGP & RIP) over different multiple-domain topologies (10 samples; confidence level: 95%).

To measure the triggered cost, we introduced random link failures, then compared the bandwidth cost of triggered SAVE updates with that of triggered routing updates; here, the routing protocols are BGP and RIP combined in multiple-domain topologies. The comparison over a specific simulated topology with a total of 90 routers and 97 links is shown in Fig. 15. Depending on the topology and the number and location of failed links, the cost varies for both SAVE and routing protocols. In most cases, however, SAVE has lower triggered bandwidth cost than routing protocols. Topology changes often start a chain reaction of triggered routing updates; by contrast, not all of these changes lead to forwarding table changes. Thus SAVE updates are not always triggered and less bandwidth is consumed.

Finally, the bandwidth cost incurred by routing protocols is already quite small compared to data traffic over the Internet. For instance, in our simulation, SAVE's bandwidth cost per link for a 92-router topology is around 120 bytes/s per link, whereas many real routers are capable of handling traffic in a

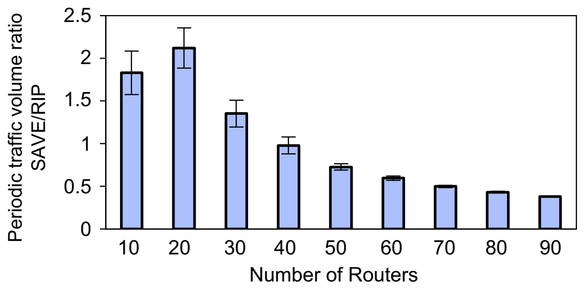


Fig. 13. Periodic bandwidth cost comparison between SAVE and RIP over different single-domain topologies (10 samples; confidence level: 95%).

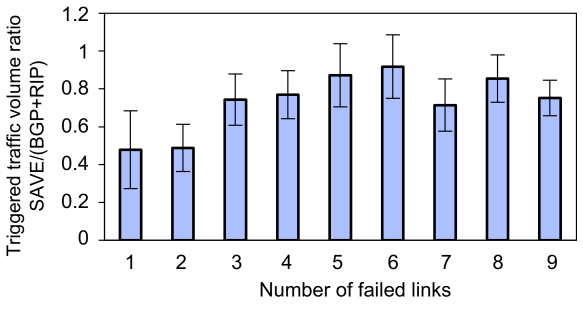


Fig. 15. Triggered bandwidth cost comparison between SAVE and routing protocols over a 97-link multiple-domain topology (10 samples; confidence level: 95%).

much greater magnitude of 10 Gbps or even 100 Tbps. Incurring a bandwidth cost of the same or less magnitude than routing protocols, SAVE only introduces a small amount of traffic into the Internet. SAVE's benefits should outweigh this cost.

9. Open issues

9.1. Securing the SAVE protocol

The SAVE protocol builds incoming tables usable for a variety of purposes, including providing security to the network. Special care must be taken to secure the SAVE protocol against malicious attempts to compromise, misuse or disable the protocol. The SAVE update exchange process between routers must be protected.

Securing the SAVE protocol shares some similarities with securing a routing protocol. Just as routing updates must be protected to allow correct routing protocol operation, SAVE updates must be protected to allow correct SAVE operation. We believe that existing and upcoming approaches to securing routing protocols can be leveraged to secure SAVE updates.

Given the above discussion, we suggest that:

- SAVE updates should be exchanged only between routers, excluding regular hosts. Thus, in order to mount an attack via SAVE updates, the attacker would need to compromise some router.
- Routers should establish trust relationships prior to exchanging SAVE updates.
- Each SAVE update should be signed (or encrypted) to guarantee its integrity. Replay of SAVE updates must also be prevented, using standard cryptographic methods.
- The processing (including the authentication) of SAVE updates should be lightweight to prevent a DoS attack on the SAVE router. If a SAVE router only communicates with trusted neighbors and can do so in a lightweight fashion, DoS attacks will have fewer chances to succeed.

The SAVE protocol also has a correctness issue similar to that of routing protocols—a compromised router, if undetected, can severely damage the proper functioning of the network by sending bogus SAVE updates. Some simple intrusion detection implemented in routers would help to counter this problem.

A compromised or improperly configured router may also allow some spoofing packets to get through. If a SAVE router's egress filtering is disabled, for instance, attackers residing within that router's local network would be able to spoof the addresses of networks upstream from that router. Note that these attackers would still not be able to send spoofed packets with any source and any destination, but only packets with those source/destination pairs for which their router is along the path.

9.2. Incremental deployment

To be of any practical use, SAVE must provide substantial value even when it is only incrementally deployed. SAVE must ensure that incoming tables can still be properly established and maintained in the presence of legacy routers, which do not run SAVE. Also, SAVE must handle data packets that carry source addresses from a legacy router's address space.

With incremental deployment, those packets which carry source addresses that cannot be found in a router's incoming table can be flagged by the router, rather than immediately dropped. This flag can tell a higher layer (such as transport or application layer) that special handling is needed. One possibility would be to deliver copies of such packets to an intrusion detection system near the target address.

If a region's routers deploy SAVE, one immediate advantage gained is that the address space of this region will be recorded in other SAVE routers' incoming tables, making the space unlikely to be chosen for spoofed source addresses. Recall that one typical DoS attack is to put the victim's address in the source address of TCP SYN packets, causing the victim to be flooded by SYN-ACK packets. Deploying SAVE routers protects the local address space from this attack.

Researchers at Purdue have evaluated incremental deployment of route-based distributed packet filtering (DPF) and suggested a deployment strategy that decreases the number of spoofable addresses while minimizing the percentage of routers performing the filtering [20]. Since route-based DPF is indeed incoming-table-based filtering, this research is complementary with the SAVE protocol and directly applicable to many aspects of SAVE's deployment. It suggests that using incoming tables created by SAVE for source address validation will work well even if only a small percentage of routers run incoming-table-based filtering. It also indicates that incoming tables built with incremental SAVE deployment can be useful for traceback.

Incremental deployment of SAVE is complex and still has open issues. We are currently in the process of enhancing SAVE for incremental deployment [43].

10. Conclusion

Up to this point packet delivery over the Internet has been solely based on destination-address-directed forwarding. Attackers have exploited this to forge source addresses in their malicious packets to disguise their identities. Yet, without the knowledge of what direction a packet should arrive from, routers cannot filter out such attack packets. Asymmetric network routing, which became common over the years, also makes it difficult for routers to obtain such incoming direction knowledge.

Today's Internet requires correct, reliable, and secure incoming direction knowledge at all routers that need it. The SAVE protocol is the first practical step in making it possible to learn the valid incoming direction of IP packets.

The incoming tree mechanism of SAVE allows every SAVE router to learn the valid incoming direction of every packet, and more importantly, to react to routing changes with a low cost and in a timely manner! Furthermore, except for certain optional optimizations, SAVE operates by assuming no specifics of routing protocols; every SAVE router only needs to consult its forwarding table. We have demonstrated that the protocol produces correct incoming tables with reasonable costs, comparable to or less than the costs of creating forwarding tables.

We believe that the benefits of incoming direction knowledge justifies the cost of running SAVE. If for no other reason, incoming tables are already of clear value in handling the prevalent use of forged IP source addresses on attack packets. Both manual and automated responses to network attacks will be easier as the defenders will have confidence whether every packet bears a correct address, or at least an address on the same network as the attacking machine.

As we continue to improve the protocol and investigate its utility, we believe that the incoming knowledge made available by SAVE will be equally useful for many other purposes. Network problem diagnosis, IP multicast routing protocols, and various source-address-based services will all benefit from SAVE.

References

[1] J. Li, J. Mirkovic, M. Wang, P.L. Reiher, L. Zhang, SAVE: Source address validity enforcement protocol, in: Proceedings of IEEE INFOCOM, New York, 2002, pp. 1557–1566.

[2] Computer Emergency Response Team, CERT advisory CA-2000-01 denial-of-service developments, http://www.cert.org/advisories/CA-2000-01.html (January 2000).

[3] C.L. Schuba, I.V. Krsul, M.G. Kuhn, E.H. Spafford, A. Sundaram, D. Zamboni, Analysis of a denial of service attack on TCP, in: Proceedings of the 1997 IEEE Symposium on Security and Privacy, 1997, pp. 208–223.

[4] Computer Emergency Response Team, CERT advisory CA-1998-01 smurf IP denial-of-service attacks, http://www.cert.org/advisories/CA-1998-01.html (January 1998).

[5] V. Paxson, An analysis of using reflectors for distributed denial-of-service attacks, ACM Computer Communications Review (CCR) 31 (3) (2001) 38–47.

[6] S.M. Bellovin, ICMP traceback messages, work-in-progress Internet Draft: draft-bellovin-itrace-00.txt, March 2000.

[7] S. Savage, D. Wetherall, A.R. Karlin, T. Anderson, Network support for IP traceback, IEEE/ACM Transactions on Networking 9 (3) (2001) 226–237.

[8] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, B. Schwartz, S.T. Kent, W.T. Strayer, Single-packet IP traceback, IEEE/ACM Transactions on Networking 10 (6) (2002) 721–734.

[9] R. Stone, CenterTrack: An IP overlay network for tracking DoS floods, in: Proceedings of the USENIX Security Symposium, 2000.

[10] A. Yaar, A. Perrig, D. Song, Pi: A path identification mechanism to defend against DDoS attack, in: Proceedings of the IEEE Symposium on Security and Privacy, 2003, pp. 93–107.

[11] A. Yaar, A. Perrig, D. Song, StackPi: New packet marking and filtering mechanisms for DDoS and IP spoofing defense, IEEE Journal of Selected Areas in Communications 24 (10) (2006) 1853–1863.

[12] F. Baker, Requirements for IP Version 4 routers, RFC 1812 (1995). http://www.ietf.org/rfc/rfc1812.txt.

[13] V. Paxson, End-to-end routing behavior in the Internet, in: Proceedings of the ACM SIGCOMM, 1996.

[14] P. Ferguson, D. Senie, Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing, RFC 2827 (2000). http://www.ietf.org/rfc/rfc2827.txt.

[15] T. Killalea, Recommended Internet service provider security services and procedures, RFC 3013 (2000). http://www.ietf.org/rfc/rfc3013.txt.

[16] S. Kent, R. Atkinson, Security architecture for the Internet Protocol, RFC 2401, obsoleted by RFC 4301, updated by RFC 3168 (1998). http://www.ietf.org/rfc/rfc2401.txt.

[17] A. Bremler-Barr, H. Levy, Spoofing prevention method, in: Proceedings of the IEEE INFOCOM, 2005.

[18] C. Jin, H. Wang, K.G. Shin, Hop-count filtering: An effective defense against spoofed DDoS traffic, in: Proceedings of the Conference on Computer and Communications Security, 2003, pp. 30–41.

[19] S.J. Templeton, K.E. Levitt, Detecting spoofed packets, in: Proceedings of the DARPA Information Survivability Conference and Exposition, vol. 1. 2003, pp. 164–175.

[20] K. Park, H. Lee, On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets, in: Proceedings of the ACM SIGCOMM, 2001, pp. 15–26.

[21] Y.K. Dalal, R.M. Metcalfe, Reverse path forwarding of broadcast packets, Communications of the ACM 21 (12) (1978) 1040–1048.

[22] S.E. Deering, D.R. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Transactions on Computer Systems 8 (2) (1990) 85–110.

[23] T. Ballardie, P. Francis, J. Crowcroft, Core based trees (CBT): an architecture for scalable inter-domain multicast routing, in: Proceedings of the SIGCOMM, 1993.

[24] S. Deering, D.L. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, L. Wei, The PIM architecture for wide-area multicast routing, IEEE/ACM Transactions on Networking 4 (2) (1996) 153–162.

[25] F. Baker, P. Savola, Ingress filtering for multihomed networks, RFC 3704 (2004). http://www.ietf.org/rfc/rfc3704.txt.

[26] Z. Duan, X. Yuan, J. Chandrashekar, Constructing inter-domain packet filters to control IP spoofing based on BGPupdates, in: IEEE Infocom, 2006.

[27] H. Lee, M. Kwon, G. Hasker, A. Perrig, BASE: An incrementally deployable mechanism for viable IP spoofing prevention, in: Proceedings of the ACM Symposium on Information, Computer, and Communication Security, 2007.

[28] S. Kent, K. Seo, Security architecture for the Internet protocol, RFC 4301 (2005). http://www.ietf.org/rfc/rfc4301.txt.

[29] H. Burch, W. Cheswick, Tracing anonymous packets to their approximate source, in: Proceedings of the USENIX LISA, 2000.

[30] M. Ma, Tabu marking scheme for IP traceback, in: Proceedings of the IPDPS, 2005.

[31] D. Dean, M.K. Franklin, A. Stubblefield, An algebraic approach to IP traceback, ACM Transactions on Information and System Security 5 (2) (2002) 119–137.

[32] M. Adler, Trade-offs in probabilistic packet marking for IP traceback, Journal of the ACM 52 (2) (2005) 217–244.

[33] M.T. Goodrich, Efficient packet marking for large-scale IP traceback, in: Proceedings of the Conference on Computer and Communications Security, 2002, pp. 117–126.

[34] A. Yaar, A. Perrig, D. Song, FIT: Fast Internet traceback, in: Proceedings of the IEEE INFOCOM, 2005.

[35] K. Park, H. Lee, On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack, in: Proceedings of the IEEE INFOCOM, 2001.

[36] J. Moy, OSPF Version 2, RFC 2328 (Standard) (1998). http://www.ietf.org/rfc/rfc2328.txt.

[37] C. Perkins, IP mobility support for IPv4, RFC 3344 (2002). http://www.ietf.org/rfc/rfc3344.txt.

[38] G. Montenegro, Reverse Tunneling for Mobile IP, RFC 2344, obsoleted by RFC 3024 (1998). http://www.ietf.org/rfc/rfc2344.txt.

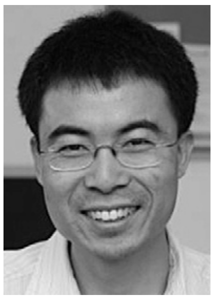
[39] D. Johnson, C. Perkins, J. Arkko, Mobility support in IPv6, RFC 3775 (2004). http://www.ietf.org/rfc/rfc3775.txt.

[40] E.W. Zegura, K.L. Calvert, S. Bhattacharjee, How to model an internetwork, in: Proceedings of the IEEE INFOCOM, 1996, pp. 594–602.

[41] Y. Rekhter, T. Li, A Border Gateway Protocol 4 (BGP-4), RFC 1771 (Draft Standard), obsoleted by RFC 4271 (1995), http://www.ietf.org/rfc/rfc1771.txt.

[42] G. Malkin, RIP Version 2, RFC 2453 (Standard) (1998), http://www.ietf.org/rfc/rfc2453.txt.

[43] T. Ehrenkranz, J. Li, An incrementally deployable protocol for learning the valid incoming direction of IP packets, Technical Report CIS-TR-2007-05, University of Oregon, March 2007.



Jun Li is an assistant professor at the University of Oregon, and directs the Network Security Research Laboratory there. He received his Ph.D. from UCLA in 2002 (with honors), M.E. from Chinese Academy of Sciences in 1995 (with Presidential Scholarship), and B.S. from Peking University in 1992, all in computer science. His current research includes Internet worm detection, BGP routing, IP source address validity, and security for peer-to-peer systems. He is a 2007 recipient of the prestigious NSF CAREER award.

Jelena Mirkovic is a computer scientist at the USC Information Sciences Institute, which she joined in 2007. Prior to this she was an assistant professor at the University of Delaware, 2003–2007. She received her M.S. and Ph.D. from UCLA, where she worked on several network security problems as a member of LASR group, lead by Prof. Peter Reiher. She received her B.S. in Computer Science and Engineering from the School of Electrical Engineering, University of Belgrade, Serbia. Her current research is focused on several important network security problems: computer worms and viruses, denial-of-service attacks, and IP spoofing.

Toby Ehrenkranz: An Oregon native, he received his B.S. in Mathematics and Computer Science from the University of Oregon in 2002. After teaching kindergarten in Chengdu, China for two years, he returned to the University of Oregon where he is currently working towards his Ph.D. His research is focused on worms and IP source address validity.

Peter Reiher received his B.S. in Electrical Engineering and Computer Science from the University of Notre Dame in 1979. He received his M.S. and Ph.D. in Computer Science from UCLA in 1984 and 1987, respectively. He has done research in the fields of distributed operating systems, security for networks and distributed computing, file systems, optimistic parallel discrete event simulation, ubiquitous computing, naming issues in distributed systems, active networks, and systems software for mobile computing. Dr. Reiher is an Adjunct Associate Professor in the Computer Science Department at UCLA.

Lixia Zhang received her Ph.D. in computer science from the Massachusetts Institute of Technology. She was a member of the research staff at the Xerox Palo Alto Research Center before joining the faculty of UCLA's Computer Science Department in 1996. In the past she has served as the vice chair of ACM SIGCOMM, Co-Chair of IEEE ComSoc Internet Technical Committee, and on the editorial board for the IEEE/ACM Transactions on Networking. She is now a member of the Internet Architecture Board, and co-chairs IRTF Routing Research Group. Her research interests includes security and dynamics in large scale systems.