SAVE: Source Address Validity Enforcement Protocol[†]

Jun Li, Jelena Mirkovic, Mengqiu Wang, Peter Reiher, and Lixia Zhang

Computer Science Department
University of California, Los Angeles

***Abstract*—Forcing all IP packets to carry correct source addresses can greatly help network security, attack tracing, and network problem debugging. However, due to asymmetries in today's Internet routing, routers do not have readily available information to verify the correctness of the source address for each incoming packet.**

In this paper we describe a new protocol, named SAVE, that can provide routers with the information needed for source address validation. SAVE messages propagate valid source address information from the source location to all destinations, allowing each router along the way to build an incoming table that associates each incoming interface of the router with a set of valid source address blocks. This paper presents the protocol design and evaluates its correctness and performance by simulation experiments. The paper also discusses the issues of protocol security, the effectiveness of partial SAVE deployment, and the handling of unconventional forms of network routing, such as mobile IP and tunneling.

I. INTRODUCTION

Ensuring that an IP packet carries a correct source address would be valuable for many purposes. Services that rely on correct source addresses (congestion control, fair queuing, source-based traffic control schemes) would profit. Network problem diagnosis, now able to locate the possible sources of a problem, could also be simplified.

Moreover, this would assist in solving one of the most important problems in network security: attackers commonly forge source addresses to avoid responsibility for their malicious packets. Examples include DDoS attacks [24], TCP SYN flooding attacks [21], and smurf attacks [23]. Reliably locating the attacker has been difficult because defenders cannot easily verify that a packet was indeed sent by the node specified in its source address.

Existing approaches to handling forged IP source addresses include:

- Tracing back the source of the forged packets from their destination with the cooperation of network routers [3] [4] [20][22]
- Filtering forged packets at the first router encountered on entering the Internet, which typically has information about valid source addresses that pass through it (*ingress filtering*) [9]
- Filtering forged packets on the basis of forwarding tables that do not take asymmetries into account [1]
- Using cryptographic authentication, such as IPsec[11]

[†] This work is funded by NSF under contract number ANI-9980501. Authors can be contacted at {lijun, sunshine, wangmq, reiher, lixia}@cs.ucla.edu.

These approaches solve part of the problem, but do not address all issues. In particular, paths through the Internet are frequently asymmetric. According to [18], a path through the Internet in 1995 visited different cities in each direction 50% of the time and different autonomous systems 30% of the time. As a result, the forwarding tables used by routers to deliver packets are not reliable for determining where packets come from.

If we had reliable tables at many routers specifying proper incoming interfaces for source addresses, an attacker's choice of forgeable IP source addresses would be sharply reduced. All improperly addressed packets could be easily dropped as soon as the forgeries were detected. Attack tracing tools could also use the knowledge produced by address validation to determine the possible sources of attacks.

This table could also be used for non-security purposes. Reverse path forwarding (RPF) would be more effective if RPF had this knowledge, for example. Multicasting protocols that use RPF to build reverse shortest-path multicasting trees (such as DVMRP [7], CBT [2] and PIM [8]) could thus build true shortest-path trees.

We present here the *Source Address Validity Enforcement* (*SAVE*) protocol. SAVE runs on individual routers and builds *incoming tables* for them, allowing each router to verify whether each packet arrives at the expected interface. Although the incoming tables built by SAVE are suitable for any of the purposes described earlier, when describing SAVE throughout this paper, we will use the example of filtering packets with forged source addresses.

The rest of this paper is organized as follows: Section II discusses the principles underlying the SAVE protocol; Sections III and IV describe the SAVE protocol in detail; Section V discusses how SAVE can be secured against attacks; Section VI presents simulation results on the costs of running the protocol and demonstrations of its efficacy; Section VII addresses the deployment of the SAVE; Section VIII discusses related work; and Section IX concludes the paper.

II. DESIGN PRINCIPLES

While a forwarding table specifies the outgoing interface for a given destination address space, an incoming table should specify the valid incoming interface for a given source address space. This similarity suggests that a simple reversal or slight modification of the existing routing protocols could yield an effective SAVE protocol. However, the information needed to construct an incoming table proves to be inherently different from that used to build a forwarding table, thus forcing a different protocol design.

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 preference rules to calculate its best outgoing interface for each destination address space.

While routing updates are used to calculate the best path, SAVE updates should be designed to inform routers about the path that has already been chosen, thus allowing all routers on the path to a destination to deduce valid incoming interfaces for specific source addresses.

In the following discussion, we further define the desirable properties of the SAVE protocol and SAVE updates.

A. *Properties of the SAVE Protocol*

The SAVE protocol is run by routers in parallel with the routing protocol. The following properties are desirable:

- *Routing protocol independence*—SAVE must be modular and independent of the underlying routing protocol, so that it can easily run on top of different routing infrastructures.
- *Immediate response to routing changes*—SAVE should respond to routing changes immediately to adjust incoming table entries.
- *Security*—SAVE must be secured or attackers could easily bypass any security it offers; worse, they could directly use SAVE for certain attacks.
- *Incremental deployment*—SAVE can only be deployed incrementally, and should offer benefits with partial deployment.
- *Low overhead*—SAVE must be lightweight in order to minimize router overhead and scale well while achieving its goals.

B. *Properties of the SAVE Updates*

The following are desirable properties of SAVE updates:

- *End-to-end communication*—SAVE updates must travel through the same routers that data packets use to reach their destination address space in order to create accurate incoming tables at those routers.
- *Aggregation of SAVE updates*—SAVE updates should be aggregated along the route as much as possible to reduce bandwidth consumption.
- *Minimized duplication*—SAVE updates should avoid duplicating any information that is already communicated via routing updates.

III. Overview Of The Save Protocol

A. *A Quick Overview*

The goal of the SAVE protocol is to build a table at each router that specifies the valid incoming interface for packets carrying a given source address. Routers use this table to filter those packets with forged source addresses.

SAVE assumes that each router is associated with a set of source addresses. All packets from this address space can only reach some set of destinations via this router. A router that forwards packets for hosts on a LAN has a source address space covering addresses of those hosts; a border router of an autonomous system (AS) handles the source address space of the whole AS (only for destinations where it acts as the exit router); and a transit router with no attached hosts has a source address space consisting of all its own IP addresses.

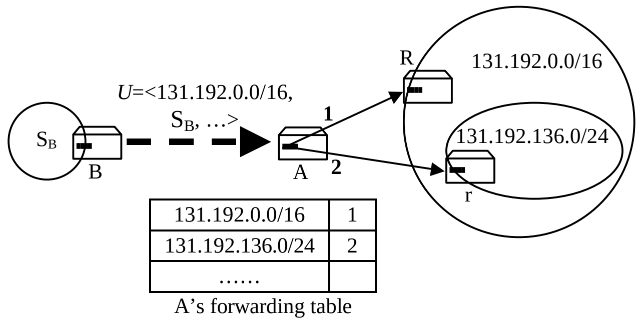
For each entry in its forwarding table, a SAVE router periodically generates SAVE updates directed toward the corresponding destination address space, in order to set up valid incoming interfaces at routers along the route. Forwarding table changes will also trigger new SAVE updates. In both cases, an update specifies the originating source address space and carries the destination address space. Since SAVE updates arrive on the same incoming interface as valid IP packets, routers between the source and final destination can record the legitimate incoming interface for the specified source address space. SAVE further allows intermediate routers to piggyback their own source address spaces on a passing-by SAVE update, thus greatly reducing bandwidth overhead.

B. *Complications*

Although the basic SAVE operations seem simple, several issues complicate the design. Here we discuss three of them: ensuring SAVE updates follow the same path as valid data packets, reacting to routing changes, and controlling SAVE bandwidth overhead.

The first issue is ensuring that the SAVE updates follow the proper paths. 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 the destination space. In Fig. 1, if router A only forwards a SAVE update toward router R, router r will not learn of the valid path for S_B. Instead, the SAVE protocol needs to generate one SAVE update toward router R, and one toward router r, to ensure proper information in incoming tables.

The second issue concerns routing changes. Routing changes establish new paths from sources to destinations that need to be validated through SAVE updates. However, not all routers that should generate SAVE updates will necessarily experience a change in their forwarding table. In Fig. 2(a), 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 Fig. 2(b). Assume that due to the failure of link BD, router D updates its forwarding table so that router C



131.192.0.0/16	1
131.192.136.0/24	2
……	

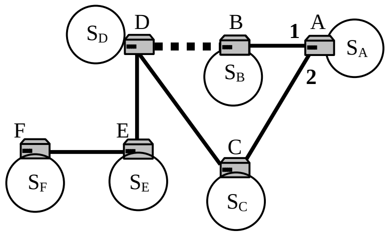
A's forwarding table

Fig. 1. An example of SAVE update forwarding.

becomes its next hop to S_A. Although D will send a new SAVE update to S_A, indicating to A that packets from S_D should now arrive from interface 2 instead, routers E and F do not change their forwarding entries to S_A and will not regenerate SAVE updates. As a result, router A will have stale information about address spaces S_E and S_F (Fig. 2(c)).

Periodically sending SAVE updates solves the problem eventually, but not quickly. SAVE handles such routing changes by employing an *incoming tree*. Each SAVE router uses SAVE updates to build the incoming tree, and then derives its incoming table from the tree. Each node on the tree represents a specific source address space and is associated with a specific incoming interface. A child inherits the same incoming interface as its parent, and thus all of its ancestors; if a node's incoming interface is changed, this change will be applied automatically to all its descendents on the tree. Consider router A in Fig. 2. Its incoming tree before link BD fails is shown in Fig. 3(a), where S_D is the parent of S_E, and S_E is the parent of S_F. Triggered by routing changes at D, S_D's new SAVE update will cause 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 map to interface 2 (Fig. 3(b)).

The third issue is overhead control. SAVE should allow an intermediate router to piggyback its own updates to



(a) A topology example
(S_X stands for router X's source address space.)

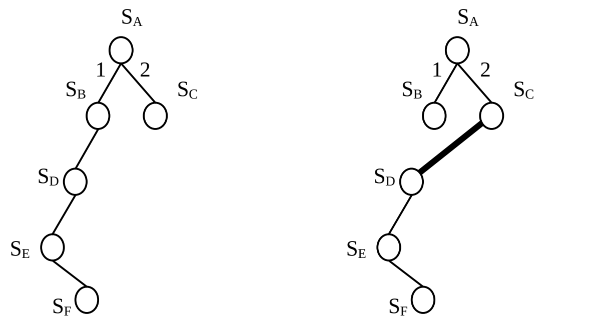
source address space	valid incoming interface
S_B	1
S_C	2
S_D	1
S_E	1
S_F	1

(b) Router A's incoming table *before* router D's routing change

source address space	valid incoming interface
S_B	1
S_C	2
S_D	2
S_E	***1*** (should be 2)
S_F	***1*** (should be 2)

(c) Router A's incoming table *after* router D's routing change

Fig. 2. An example topology of routers and their source address spaces.
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.



(a) The incoming tree at router A *before* router D's routing change

(b) The incoming tree at router A *after* router D's routing change

Fig. 3. Incoming tree example for topology in Figure 1.

SAVE updates passing through, but only if the router has not already sent its own update. This complicates the protocol, and requires that a SAVE update be marked to indicate whether it is appendable or not.

IV. Protocol Description

In this section we describe the SAVE protocol. As depicted in Fig. 4, the main components of the protocol are generating SAVE updates, processing SAVE updates, and updating the incoming tree and incoming table based on SAVE updates. We describe SAVE's key data structures and then describe each of the three operations.

A. SAVE Data Structures

SAVE employs three main data structures: the incoming table, the incoming tree, and the SAVE update. Each SAVE router has an incoming table and an incoming tree, and SAVE routers exchange SAVE updates.

The incoming table is maintained at each SAVE router and contains entries that specify a valid incoming interface for a specific source address space.

The incoming tree is maintained at each SAVE router and is used to derive the incoming table. Each tree node specifies a source address space and is mapped to the valid incoming interface for that address space. This tree structure has the following properties:

1. If a SAVE update crosses router A and then router B before reaching a router R, on R's incoming tree node S_A will be the child of node S_B. Here S_A and S_B are A's

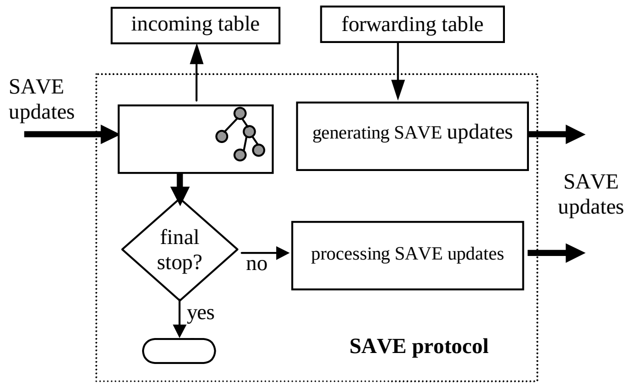


Fig. 4. The architecture of the SAVE protocol.

and B's source address space, respectively.

2. Each sub-tree directly below the tree's root is associated with an incoming interface. All nodes in a sub-tree map to the same incoming interface. This property makes building an incoming table straightforward.

Each SAVE update has three fields. The *destination address space* field specifies the final destination address space of this SAVE update. The *address space vector* (*ASV*) records source address spaces on the path that this SAVE update has traversed. The *appendable* flag indicates whether this SAVE update can have more information appended.

B. Generating SAVE Updates

A router generates SAVE updates for each 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: < destination address space = D, ASV = <S_R>, appendable = true >.

This SAVE update will be forwarded along the outgoing interface specified in the forwarding entry for D. Note that a SAVE update will be encapsulated inside an IP datagram whose destination address is randomly chosen from D, allowing routers not running SAVE to still forward SAVE updates. (See [13] for a further discussion of compatibility with legacy routers.)

Each SAVE update will cross a series of SAVE routers, each of which will update its incoming tree (and thus incoming table) based on the ASV contained in this SAVE update. The ASV itself will also be updated in transit if it is appendable, as illustrated later.

Like soft state routing protocols, SAVE supports triggered updates (when forwarding table entries change) and periodic updates. The pseudocode in Appendix I.A describes detailed steps for generating SAVE updates.

C. 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 (see Appendix I.B for a pseudocode description of the algorithm).

The ASV field records the path that the SAVE update has traversed. Because its purpose is to piggyback address space information onto a SAVE update, an ASV records an ordered list of address spaces, not a list of routers. Initially, the ASV in a SAVE update contains only the origin router's source address space. The ASV expands as the SAVE update crosses intermediate routers; an intermediate router can append its address space to the SAVE update's ASV.

In general, if an ASV has the form <S_1, S_2, …, S_n>, where S_i is the source address space of a SAVE router R_i, the SAVE update must have originated from R_1 and then crossed SAVE routers R_2, R_3…, and R_n consecutively (adjacent routers in this sequence need not be physically adjacent). Any IP packets from address space S_i will cross R_{i+1}, R_{i+2}, …, and R_n (and perhaps other routers beyond R_n) to reach the destination. (ASV maintenance is discussed further in Section IV.D.)

To preserve the properties of the tree discussed in Section IV.A, 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 ensure that this branch will map to the incoming interface that the SAVE update arrived on. Third, the procedure must ensure that information regarding those nodes on the tree that were descendents of S_i will be updated to reflect information for node S_i; the incoming interfaces of those nodes depend entirely on how R_i will forward their IP packets to reach this router.

The tree update procedure therefore parses the ASV in reverse order (see Appendix I.B), processing the last ASV element S_n 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 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 relocation could map the S_i sub-tree to a new interface.

A SAVE router might receive two SAVE updates from different incoming interfaces concerning the same source address space. We solve this problem by prioritizing SAVE updates (refer to [13] for details).

D. Processing SAVE Updates

Upon receipt of a SAVE update, a SAVE router will first use the update to maintain its incoming tree and table. After that, this update is further processed to help downstream routers maintain their trees and tables. SAVE update processing ensures that the SAVE protocol as a whole achieves two important goals:

1. Recording the path that the SAVE update has traversed before reaching a SAVE router
2. Assuring that the SAVE update follows the same path toward the specified destination address space as do valid data packets

Items 1 and 2 are addressed below in Section IV.D.1 and Section IV.D.2, respectively. (Refer to Appendix I.C for the pseudocode description.)

D.1 Maintaining the address space vector

As stated previously, the ASV field of a SAVE update records the path that the update has traversed. However, an ASV does not necessarily record the complete path. If a router has just initiated a SAVE update toward the same destination address space as a passing-by SAVE update, it still appends its own source address space; however, it marks the *appendable* field in the passing-by update as "not appendable." Thus, all downstream routers will stop recording their source address spaces into the ASV, but they will still be able to obtain the complete path information by combining ASVs from multiple updates, as illustrated below.

Assume a downstream router R receives a SAVE update that originated from R_1 (thus called R_1's SAVE update). Its ASV is expressed as <S_1, S_2, …, S_n>, where S_i is the source address space of a SAVE router R_i. If R_n is the SAVE router immediately preceding R, R_1's SAVE update to R contains the complete path information. If that is not the case, the ASV suggests that an intermediate router R_n has already initiated a SAVE update toward the same destination address space. In this case, recording the path information beyond R_n in R_1's SAVE update is redundant, since R has already obtained this information from R_n's SAVE update, whose ASV is $< S_n, S_{n+1}, \ldots >$. Any downstream SAVE router after R_n, including R, can obtain the complete path information by concatenating the ASV of R_1's SAVE update and the ASV of R_n's SAVE update. (This can happen recursively on R_n's SAVE update.) Note that this concatenation does not happen literally; instead, it is implicit because of the incoming tree update procedure.

D.2 Forwarding SAVE updates

To reach its destination address space, denoted as ***D***-space, a SAVE update must follow the same path as valid data packets toward ***D***-space. Following the same path ensures that the incoming interface of the SAVE update is the same as the incoming interface for all source address spaces carried in its ASV field. Thus, regular IP forwarding tables are used by intermediate routers to forward SAVE updates.

A problem arises when a router does not have a single forwarding entry that points exactly to the ***D***-space. Due to forwarding table aggregation, a router could have a forwarding entry for a sub-area of ***D***-space, or an entry for a superset of ***D***-space. SAVE handles this as follows:

- For each forwarding entry that specifies a route toward a sub-area of ***D***-space, a new SAVE update will be created, 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-area. The new update is forwarded further according to the corresponding forwarding entry.
- If the combined subset-type forwarding entries do not cover the whole ***D***-space, the smallest superset forwarding entry will also be used to forward the SAVE update, since it would be used for forwarding the valid data packets toward the uncovered part of ***D***-space.

SAVE updates are forwarded downstream until they reach the SAVE router that can handle all IP addresses in the destination address space without further forwarding

D.3 Overhead control

Just as source address spaces should not be appended to a passing-by update's ASV if an update has already been sent for them, updates should not be sent for address spaces that have been appended to the ASVs of passing-by updates, thus avoiding further overhead. SAVE also does not forward *replaceable* updates. An update is replaceable from the point of view of a specific SAVE router if each 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.

E. Correctness of SAVE Address Validation

SAVE's guarantee of source addresses validity is not absolute. If an incoming table indicates that a range of IP addresses comes in on a particular interface on the router, the router itself has no way of knowing if one of the machines with an address in that range forged the source address of another machine in the same range. In a simple case, a router attached to an Ethernet could not use SAVE methods to detect one machine on that Ethernet forging the address of another. The same observation applies to using SAVE for other purposes.

V. 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 is similar to 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 updates 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 kind of simple intrusion detection implemented in routers might help to counter this problem.

VI. SIMULATION

A. Simulation Design

The SAVE protocol has been implemented and tested in a custom simulation environment. While we are also working on SAVE's actual implementation and evaluation in a testbed environment, simulation is the best approach to verify our design early on and to investigate SAVE's scaling properties, which would be hard to assess in a limited testbed environment.

The goal of the simulation is to verify the effectiveness of SAVE, analyze the transient behavior of SAVE, and evaluate the bandwidth and storage 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 [5] to generate inter-domain connectivity and intra-domain connectivity. We simulated BGP [19] for inter-domain routing and RIP [15] for intra-domain routing. We also introduced asymmetric routing. The BGP simulation implements the routing policy recommended by Cisco [14].

B. *Simulation Results*

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.

B.1 *Effectiveness verification*

To verify the effectiveness of the SAVE protocol, 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

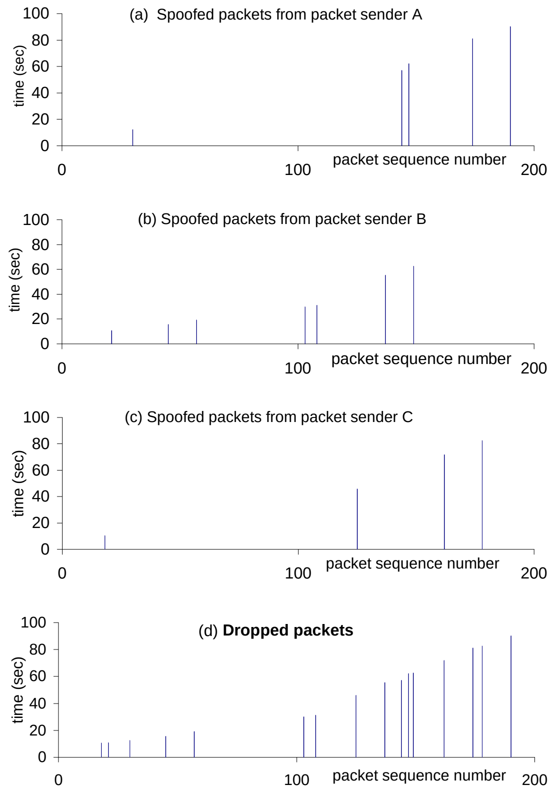


Fig. 5. SAVE effectiveness verification.

In this scenario, a DDoS attack is performed from three different machines. Every packet in the simulation has a global unique sequence number. Figures (a), (b) and (c) show the pattern of spoofed packets generated at three machines (legitimate packets are not shown). Figure (d) shows the patterns of packets that are dropped by SAVE. We can observe that all spoofed packets are dropped.

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 is effective, all spoofed packets should be caught and dropped, leading to a distribution pattern over time of dropped packets matching the traffic 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. 5.

B.2 *Transient behavior of SAVE*

When a forwarding table changes and a new route to a destination address space is 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 can be erroneously dropped even though it carries an authentic 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 only be dropped by mistake if:

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 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 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.

B.3 *The cost of the SAVE protocol*

We measured the bandwidth and storage costs of the SAVE protocol versus those of routing protocols. (Theoretical cost analysis and comparison are further given in [13].)

To measure the storage cost, we compared the size of the corresponding fast-path data structures: the incoming table used by SAVE and the forwarding table used by RIP or BGP. Fig. 6 compares SAVE with RIP for different single-domain topologies. Fig. 7 compares SAVE with BGP for different multiple-domain topologies.

The incoming table can be optimized to reduce the storage cost by leveraging symmetries in network routing. If the valid incoming interface for a specific address space is exactly the same as the outgoing interface to reach that address space, the forwarding table entry that points to this address space can be used to derive the incoming interface and validate the source address. Otherwise, a flag can be added to the forwarding table entry to indicate that the

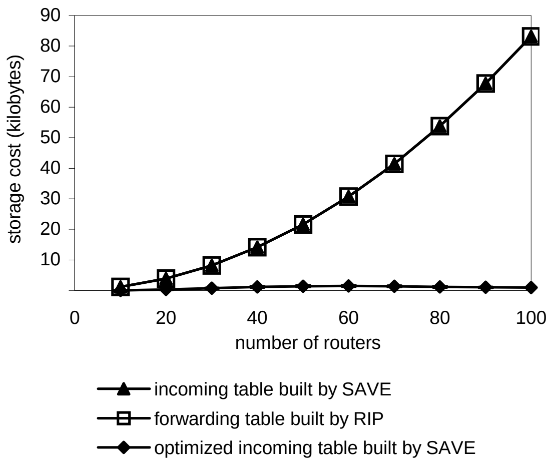


Fig. 6. Storage cost comparison for single-domain topologies.

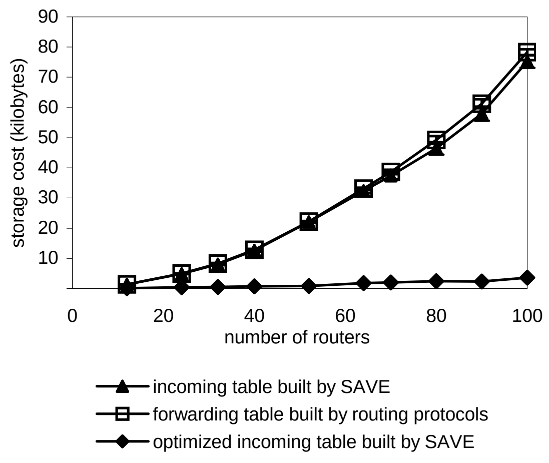


Fig. 7. Storage cost comparison for multiple-domain topologies.

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 asymmetry present. Fig. 6 and Fig. 7 show that for reasonable cases the storage cost of the optimized incoming table can be minimal.

To assess the bandwidth requirements of SAVE, we compared the triggered and periodic bandwidth cost between SAVE and routing protocols.

Assuming SAVE updates and routing updates are initiated with the same frequency, we compared SAVE and RIP in terms of periodic bandwidth cost over single-domain topologies. Simulations over different topologies show similar results (Fig. 8), 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. 6, the maximum per-link bandwidth cost varies from 3.2 to 6.9 kilobytes/sec.

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. 9. SAVE uses less than 60% of the bandwidth of the BGP and RIP combined, in the worst case measured. The maximum per-link bandwidth cost here varies from 0.6 to 6.4 kilobytes/sec over the topologies we used in Fig. 7.

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. 10. 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.

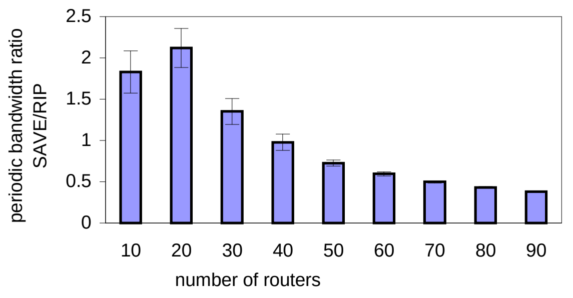


Fig. 8. Periodic bandwidth cost comparison between SAVE and RIP over different single-domain topologies (confidence level: 95%).

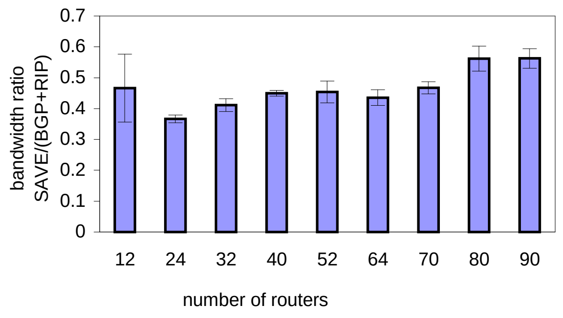


Fig. 9. Bandwidth comparison between SAVE and routing protocols (BGP &RIP) over different multiple-domain topologies (confidence level: 95%).

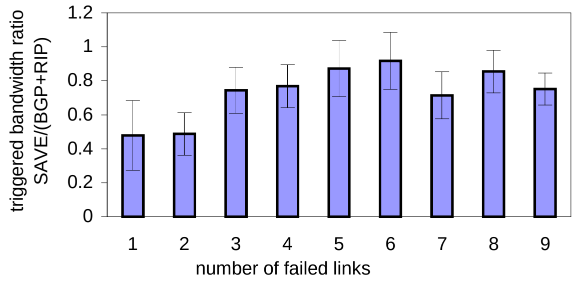


Fig. 10. Triggered bandwidth cost comparison between SAVE and routing protocols over a 97-link multiple-domain topology (confidence level: 95%).

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/sec per link, whereas many real routers are capable of handling traffic in a 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.

VII. Deployment of the SAVE Protocol

A. Partial Deployment

To be of any practical use, SAVE must provide substantial value even when it is only partially 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 partial 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 partial 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 [17]. 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 partial SAVE deployment can be useful for traceback.

Partial deployment of SAVE is complex and still has many open issues. Research continues on this problem.

B. 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 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 [16], 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, 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 at the ingress of a tunnel before a packet enters 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 fundamentally bypasses the routing protocol (source routing has the same problem). To accommodate this case, routers at the endpoints of a tunnel can send out special-purpose SAVE updates to build correct incoming table entries.

VIII. Related Work

Much network security research has focused on applying cryptographic operations in order to guarantee authenticity of packet information. IPsec is one representative at the IP layer [11]. A packet's authenticity can be guaranteed by signing or encrypting it. However, the high computation overhead of cryptographic operations prevents such approaches from being widely employed per packet. The hop integrity approach proposed in [10] uses a lighter-weight signing technique, but it has to be deployed on a per-hop basis; thus each router that needs to forward a packet must incur extra overhead for cryptographic operations.

Other research addresses IP spoofing through both preventive approaches and reactive approaches. Filtering is a preventive approach. Tracing is primarily reactive.

Reference [1] proposes a general filtering approach where many fields, including but not limited to source address, can be used for filtering. Martian address filtering is required in order to discard packets if their source addresses are special addresses (loopback address, broadcast address, etc.) or are not unicast addresses.

Forwarding-table-based-filtering is one possible approach to validating source address. It 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, invalidating this assumption and causing many legitimate packets to be dropped. As a result, this feature is often disabled since it leads to erroneous packet dropping when asymmetric paths are used.

Ingress filtering proposed in [9] ensures that packets leaving the domain of a periphery router have a source address from inside the domain, and packets entering have an outside source address, effectively providing a special-purpose incoming table only at network ingress. However, research has shown that unless ingress filtering is deployed almost everywhere, nearly arbitrary forgery is still possible [17]. Further, this approach offers no help in providing address assurance for any other purposes.

Route-based distributed packet filtering suggested in [17] studied benefits of such filtering for attack prevention and traceback, and partial deployment strategies (Section VII.A).

Packet tracing has been widely studied [3] [4] [20] [22]. 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 traveling path [20]. Tracing becomes ineffective when the volume of attack traffic is small or the attack is distributed [12]. Moreover, tracing is typically performed after an attack is detected, possibly too late to avoid damage.

Network intrusion detection has also studied approaches to localize an attacker. For instance, DECIDUOUS dynamically builds IPsec security associations to reveal the location of attacking sources [6]. However, to do this a victim running DECIDUOUS must detect the intrusion first. Network topology information is also required.

IX. 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 source address a packet should carry when it arrives, routers cannot filter out attack packets. Asymmetric network routing, which became common over the years, also raised the need for an incoming address table in order to support IP multicast routing protocols.

Today's Internet requires correct, reliable, secure incoming tables at all routers that need them. The SAVE protocol is the first practical step in making it possible to build such tables. We have demonstrated that the protocol produces correct incoming tables at reasonable costs, comparable to or less than the costs of creating routing tables. We believe that the functionality of incoming tables justifies this cost.

If for no other reason, incoming tables are already of clear value in handling the alarmingly rapid growth in the use of forged IP source addresses on attack packets. Both manual and automated responses to network attacks will be easier if the defenders have confidence that the packets bear a correct address, or at least an address on the same network as the attacking machine. The incoming tables built by the SAVE protocol can offer such assurance.

We believe that the incoming tables built by SAVE will be equally useful for many other purposes, some of which cannot be foreseen today. We will continue to improve the protocol and investigate its utility.

ACKNOWLEDGMENT

The authors would like to thank Mark Yarvis, Gregory Prier, and Janice Wheeler for their input on various aspects of this work. Our thanks also to the anonymous reviewers of this paper.

REFERENCES

[1] F. Baker. "Requirements for IP Version 4 Routers," RFC 1812, June 1995.

[2] A Ballardie, P. Francis, and J. Crowcroft. "Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing," *Proceedings of ACM SIGCOMM 1993*.

[3] S. M. Bellovin. "ICMP Traceback Messages," Internet Draft: draft-bellovin-itrace-00.txt, March, 2000.

[4] H. Burch and W. Cheswick. "Tracing Anonymous Packets to Their Approximate Source," *Proceedings of 2000 Systems Administration Conference*, December 2000.

[5] K. L. Calvert, M. B. Doar, and E. W. Zegura. "Modeling Internet Topology." *IEEE Communications Magazine 35, 6 June 1997.*

[6] H. Y. Chang, R. Narayan, S. F. Wu, B. M. Vetter, X. Wang, M. Brown, J. J. Yuill, C. Sargor, F. Jou, and F. Gong. **"**DECIDUOUS: decentralized source identification for network-based intrusions," *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, May 1999.

[7] S. E. Deering and D. R. Cheriton. "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Transactions On Computer Systems*, Vol. 8, No. 2, May 1990.

[8] S. E. Deering, D. L. Estrin, D. Farinacci, V. Jacobson, C. –G. Liu, and L. Wei. "The PIM Architecture for Wide-Area Multicast Routing," *IEEE/ACM Transactions on Networking*, Vol. 4 No. 2. April 1996.

[9] P. Ferguson and D. Senie. "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," RFC 2827, May 2000.

[10] M. G. Gouda, E. N. Elnozahy, C.–T. Huang, and T. M. McGuire. "Hop Integrity in Computer Networks," *Proceedings of the 8th IEEE International Conference on Network Protocols*, Osaka, Japan, November 2000.

[11] S. Kent and R. Atkinson. "Security architecture for the Internet protocol," RFC 2401, November 1998.

[12] H. Lee and K. Park. "On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack," *Infocom 2001*, Anchorage, Alaska, April 2001.

[13] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. "SAVE: Source Address Validity Enforcement Protocol," UCLA Technical Report 010004, 2001.

[14] B. Halabi. *Internet Routing Architectures*, Cisco Press, 1997.

[15] G. Mlakin. "RIP Version 2," RFC 2453, November 1998.

[16] G. Montenegro. "Reverse Tunneling for Mobile IP," RFC 2344, May 1998.

[17] K. Park and H. Lee. "On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets," *Proceedings of ACM SIGCOMM 2001*.

[18] V. Paxson. "End-to-End Routing Behavior in the Internet," *Proceedings of ACM Sigcomm, 1996.*

[19] Y. Rekhter and T. Li. "A Border Gateway Protocol 4 (BGP-4)," RFC 1771, July 1994.

[20] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. "Practical Network Support for IP Traceback," *Proceedings of ACM SIGCOMM 2000*.

[21] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. "Analysis of a denial of service attack on TCP," *Proc. of IEEE Symposium on Security and Privacy*, 1997.
[22] R. Stone. "CenterTrack: An IP Overlay Network for Tracking DoS Floods," *9th USENIX Security Symposium*, August 2000.
[23] 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.
[24] Computer Emergency Response Team. "CERT Advisory CA-2000-01 Denial-of-Service Developments," http://www.cert.org/advisories/CA-2000-01.html, January 2000.

APPENDIX I. SAVE PROTOCOL PSEUDOCODE DESCRIPTION

A. SAVE Update Generation Procedure

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

1   Iterate through the forwarding table
2   loop:
    for each forwarding entry $e$:
        <destination prefix, outgoing interface $oif$>
3       if (should_generate_SAVE_update_for($e$))
4           compose SAVE update $U$:
              $U \leftarrow$ <destination prefix, ASV=<$S_R$>,
                    $appendable$=true>
5           send $U$ out along interface $oif$
6   goto loop
```

B. Incoming Tree Update Procedure Upon Receipt of a SAVE Update

```
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=<$S_1$, $S_2$, …, $S_n$>
    $iif$:  the incoming interface that $U$ arrives on
    subtree(X): a sub-tree of the incoming tree that is
                rooted at X
1   [Initialization when router R boots up]
      The tree only contains the root node that
      represents $S_R$

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

/* now handle $S_{n-1}$, $S_{n-2}$, . . ., $S_2$, $S_1$ one by one */
9    for (i ← n-1; i > 0; i-- )
10     if ($S_i$ does not exist in the incoming tree)
11         graft $S_i$ under $S_{i+1}$
12     else
13         graft subtree($S_i$) directly under $S_{i+1}$ (if not)
14   end
```

C. The Processing Procedure of a SAVE Update

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

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

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

5   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 subset-type forwarding entries */
6   for every $e_i$ in $E$
7       create a SAVE update:
          $U_i \leftarrow$ <$S_{Di}$ , ASV, $appendable$>
8       processUpdate($U_i$)  /* process $U_i$ */
9   end loop

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

/* find superset-type forwarding entry with least
   coverage of $S_D$*/
13  find $f$: <$S_{D'}$, $oif'$> that $S_{D'} \supseteq S_D$ &&
                    $\neg\exists$ $e_j$=<$S_{Dj}$, $oif_j$>: $S_{D'} \supset S_{Dj} \supseteq S_D$
14  if ($f$ is not found)
      return

15  if ( $appendable$) {
16    ASV ←<ASV, $S_R$>
          /* append $S_R$; ASV=<$S_1$, $S_2$, …, $S_k$, $S_R$> */
17    if ( R has processed $f$, i.e.already generated SAVE
          update for $f$)
18      appendable ← false
19  }

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