SAVE: Source Address Validity Enforcement Protocol

Jun Li Jelena Mirkovic Mengqiu Wang Peter Reiher Lixia Zhang

ABSTRACT

Many network attacks forge the source address in their IP packets to block traceback. Recently, research activity has focused on packet-tracing mechanisms to counter this deception. Unfortunately, these mechanisms are either too expensive or ineffective against distributed attacks where traffic comes from multiple directions, and the volume in each direction is small.

We believe that the fundamental solution to the problem of source address forging is to validate source addresses throughout the network. We have developed a source address filtering protocol that establishes and maintains valid incoming interface information on source addresses at each router, thus allowing all packets carrying improper source addresses to be immediately identified. Our protocol works correctly in the presence of asymmetric routing. We will describe the protocol that gathers the information to validate source addresses and use simulation to demonstrate that it is effective and has reasonable costs.

Keywords: IP spoofing, DDoS, filtering, security

1 INTRODUCTION

Attackers commonly forge source addresses to hinder tracing of their malicious packets. Examples include DDoS attacks [32], smurf attacks [31], and TCP SYN flooding attacks [24]. Reliably detecting the attacker is hard because standard routers cannot verify that a packet was indeed sent by the node specified in its source address.

Periphery filtering is widely used to validate source addresses [12]. A periphery router ensures that a packet leaving its domain has a source address from inside the domain, and a packet entering has one from outside; but unless periphery filtering is deployed everywhere, nearly arbitrary forgery is still possible. For example, in Figure 1 an attacker in network S_M can send packets into network S_B with source address from network S_A, even though both A and B support periphery filtering.

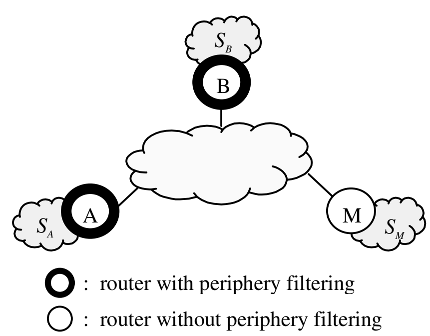


Figure 1: Periphery filtering

A router can check a packet's incoming interface[1] with *forwarding-table-based filtering* [1], where a packet is expected to arrive through the same interface that is used to send packets back to the source. Unfortunately, this does not work when asymmetry exists. If the incoming interface for an address is different from the outgoing interface for that address, valid packets from that address will be dropped. For instance, if routing between A and B in Figure 1 is asymmetric, packets from network S_A will be dropped by router B. According to [17], a path through the Internet in 1995 visited at different cities in each direction 50% of the time, and different autonomous systems 30% of the time. Asymmetry in the Internet is common, not exceptional, so filtering must account for it.

One approach to the problem of IP spoofing is tracing. Since source addresses are unreliable, tracing requires expensive and complicated techniques to observe traffic as they pass through routers and reconstruct a packet's travel path at the end. Tracing also becomes ineffective when the volume of attack traffic is small or the attack is distributed. Moreover, tracing is typically performed after an attack is detected, and perhaps the victim has already been damaged. Since tracing usually already needs to add new functionalities to routers to observe or mark traffic, we believe the most valuable functionality to add is one that will directly prevent IP spoofing.

We propose *incoming-table-based filtering* to filter packets that carry forged source addresses. In this approach, a router on the Internet builds an incoming table that specifies the correct incoming interface for a given source address, even with asymmetric routing present. When a packet arrives on an interface, a router can consult its incoming table to determine whether this packet comes from the proper direction.

Apart from IP spoofing prevention, source address validation has many other advantages. Attack tracing tools can use the knowledge of address validation and routers that perform it to narrow the possible sources of an attack. Intrusion detection and network problem diagnosis can also be simplified. Services that rely on accurate source addresses (congestion control, fair queuing, source-based traffic control schemes) also profit. Reverse path forwarding (RPF) can be more effective; multicasting protocols that use RPF to build reverse shortest-path multicasting trees (such as DVMRP [8], CBT [2] and PIM [9]) can thus build true shortest-path trees.

This paper describes a protocol used to build and maintain an incoming table and the philosophy underlying the design. We call this protocol the *source address validity enforcement protocol (SAVE)*. It can be deployed on routers running different routing protocols with reasonable cost. The protocol is described in Section 2. Section 3 discusses advanced issues, including compatibility with legacy routers, soft state maintenance, overhead control,

[1] Incoming and outgoing interfaces of a router can be physical network interfaces identified with a link-layer address, or logical network interfaces identified with a unique IP address.

and interaction of filtering with some special cases, such as mobile IP and IP multicast. Section 4 presents simulation results on the costs of running the protocol and demonstrations of its efficacy, and Section 5 discusses related work. Future work is discussed in Section 6, and we conclude in Section 7.

2 THE DESIGN OF THE SAVE PROTOCOL

2.1 Overview

The goal of the SAVE protocol is to build a table at each participating router that indicates the router's proper incoming interface for packets from all sources. The router will use packet source addresses to index the table, dropping packets that come in on interfaces not matching the table entries.

One might think that building an incoming table is conceptually the reverse of building a forwarding table, and thus a minor alteration to existing routing protocols, but actually the tasks prove very different. SAVE needs a greater knowledge of other routers' behavior than standard routing protocols require. Figure 2 shows an example. After route calculation, router 1 knows that there are two equal-cost paths from router 6 to itself. If router 1 only has knowledge of its neighborhood, it cannot determine the incoming interface for packets from 6 which could be arbitrarily far away. Router 1 needs to know how 6 breaks routing ties. Assuming 6 prefers the lower address, 1 still needs to determine which path from 6 to 1 starts with a router of lower address. In Figure 2 (a), a packet from 6 to 1 arrives via 3; and in (b), due to a difference in the upstream topology, it arrives via 2.

SAVE builds the incoming table at each router in a distributed fashion, using information in a router's forwarding table to signal to other routers the proper packet paths. SAVE must determine which paths other routers have chosen to reach all destinations. Each router sends *SAVE updates* to all destinations in its forwarding table, sending a new update when routing to a destination is changed. SAVE updates traverse the same paths as normal IP packets traverse. Each router in the path records the incoming interface used by the SAVE update as the legitimate interface for packets from upstream routers. Once all routers have sent such SAVE updates to all their destinations, each router will have a complete set of legitimate sources for each incoming interface. This information can be used to build an incoming table.

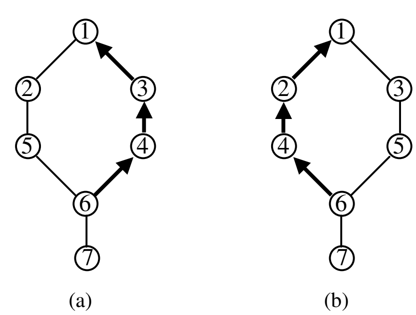


Figure 2: An example topology with two equal-cost paths

Merely keeping a list of interfaces and corresponding addresses is insufficient. If routing changes alter a source/destination path at an intermediate router, the source router might not change its next hop interface, so that router will not generate SAVE updates for its address space. Organizing SAVE's information in an incoming tree solves this problem. The tree structure stores the upstream router's address space as a descendent of the intermediate router's address space, so a change in intermediate router's interface automatically changes the upstream router's interface. For example, in Figure 2 router 7 delivers packets to router 1 through router 6, so changing the delivery path for packets from router 6 to router 1 also changes the delivery path from router 7 to router 1, even though router 7 has not changed its routing information. By using a tree for 1's information, where 7's address space is the child of 6's address space, the update that changes the interface used for router 6's address space will also change the interface used for 7's address space.

Since all of the Internet's multiple routing protocols produce a forwarding table, we avoid developing multiple versions of SAVE by working with the common forwarding table. Topology factors like node or link failure and routing policies are automatically handled by underlying routing protocols. SAVE extracts its update information from each router's forwarding table, and any changes to the forwarding table trigger new SAVE updates.

Deployment of SAVE-enabled routers will be incremental, so new routers must coexist with legacy routers. A neighboring legacy router will not help establish the incoming table of a SAVE-enabled router, except by forwarding control messages that can be treated as IP packets. SAVE is designed with this constraint in mind.

Ultimately, this protocol must work at Internet scale. Like routing protocols, the scaling factors are related to IP address space size and number of routers that must run the protocol. This paper analyzes the basic scaling costs of the SAVE protocol. A future version of the protocol will further improve SAVE's scalability through more address space aggregation. Similarly, since the purpose of this protocol is to defeat attacks, the protocol itself must be secure from attacks to offer any benefit. We do not discuss security issues in detail here, but touch upon them in the future work section.

2.2 Protocol Description

In this section we describe the SAVE protocol. We illustrate the formation and adjustment of the incoming tree and the creation of the incoming table at a router. We also describe the generation of SAVE updates at origin routers and their handling at intermediary routers. The structure of the protocol is outlined in Figure 3.

2.2.1 *Assumptions*

For ease of understanding and discussion, we make the following assumptions. Assumption (a) and (b) list only the properties of a router required by SAVE, separating each router from the specific routing protocols that it runs. Assumption (c), (d) and (e) are not mandatory and their relaxation will be discussed later. (In particular, we will address compatibility with legacy routers in Section 3.1, reliability in Section 3.2, and security in Section 6.)

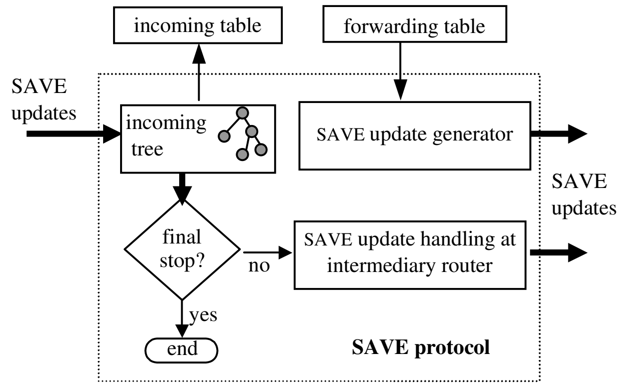


Figure 3: The architecture of the SAVE protocol

(a) Each router has a forwarding table with each entry in the form <*prefix*, *out_if*> that specifies *out_if* as the outgoing interface for a particular address space *prefix*.

(b) Each router is associated with a source address space; packets from this space reach the outside world via this router. (Note: this router is not necessarily the first hop to reach outside; for example, the default exit border router of an autonomous system (AS) can regard the whole AS as its source address space. We assume that an independent procedure exists for determining source address spaces.)

(c) Every router runs the SAVE protocol.

(d) SAVE updates between routers are reliable; they are never erroneous, lost, duplicated or out of order.

(e) SAVE updates between routers are secure.

2.2.2 *Generation of SAVE Updates*

SAVE updates are generated when the system is initialized and when changes in a router's forwarding table occur (Figure 4). A state is assigned to each forwarding entry: a newly added or updated forwarding entry is set to state *todo* and a processed one is set to state *done*. A router's SAVE updates are generated by iterating through its forwarding table. A SAVE update is created for each forwarding entry <*S*, *out_if*> in the *todo* state and sent out along *out_if* towards *S* inside an IP datagram. (The handling of a removed forwarding entry will be addressed in Section 3.2.)

Procedure: SAVE update generation at router R.

1 S_R : the address space associated with router R

2 [Initialization]
reset the state of each forwarding entry *e*: state (*e*)←*todo*

3 Iterate through the forwarding table
4 loop: **for** each forwarding entry *e*: <*S*, *out_if*>
5 **if** (state(*e*) is *todo*)
6 compose SAVE update *F*:
F ←<*S*, ASV=<S_R>, *a*=1>
7 send *F* out along interface *out_if*
8 state(e) ← *done*
9 **goto** loop

Figure 4: SAVE update generation procedure

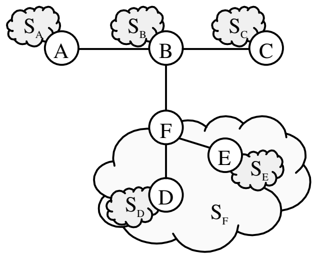


Figure 5: An example topology used in SAVE protocol description

Table 1: A partial snapshot of forwarding tables at routers in Figure 5 and the corresponding SAVE updates

A:	*	i_{AB}	<*, <S_A>, 1>
B:	S_F	i_{BF}	<S_F, <S_B>, 1>
C:	S_F	i_{CB}	<S_F, <S_C>, 1>
D:	*	i_{DF}	<*, <S_D>, 1>
E:	*	i_{EF}	<*, <S_E>, 1>
F:	S_D	i_{FD}	<S_D, <S_F>, 1>
	S_E	i_{FE}	<S_E, <S_F>, 1>
	*	i_{FB}	<*, <S_F>, 1>

A SAVE update contains a destination address space *S*, an *address space vector ASV*, and a flag *a*. The flag *a* indicates whether more information should be appended to the update along its way toward the destination (to be discussed in Section 2.2.4). When a SAVE update is initiated, its ASV only contains one element—the source address space associated with the origin router.

Table 1 illustrates the SAVE update generation for the topology shown in Figure 5. This topology has six routers A through F, each having an associated address space S_A through S_F, respectively. i_{XY} denotes the interface of X that has a direct link with router Y (X or Y=A, B, C, D, E, F). S_F includes S_D and S_E. Table 1 shows a partial snapshot of forwarding tables that are relevant to reaching F and the corresponding SAVE updates.

2.2.3 Incoming Tree Creation and Maintenance

The incoming tree at a router maintains the information about valid interface for every source address. It has two aspects. (1) Each node on an incoming tree represents an address space. On router R's incoming tree, a node for address space *A* will be a child of a node for address space *B* if packets from *A* must cross *B* to reach R; the root of the tree is the source address space of R. For a given node on the tree, its path to the root corresponds to a sequence of address spaces crossed to reach R. (2) Each node on the tree maps to an incoming interface. All nodes of a sub-tree directly under the root will be associated with the same incoming interface.

An incoming table can be easily constructed from an incoming tree. Nodes with same interface may be further aggregated. The table's data structure can also be designed to achieve the best efficiency for validating source addresses of packets.

Each SAVE update carries an ASV: <S_1, S_2, …, S_n>. When this update is received at router R, its ASV indicates that packets from address space S_i (i=1, 2, …, n-1) will cross S_{i+1}, S_{i+2}, …, and S_n, and perhaps other address spaces after S_n, to reach R.

A SAVE update alters a router's tree (Figure 6). Its ASV is parsed in reverse. If the last ASV element S_n does not exist in the incoming tree, it will be grafted directly under the root; if the current interface bound with S_n is not *F*'s incoming interface, the S_n sub-tree will be remapped to the new interface and grafted under the root. Any other element of ASV, S_i (i≠n), and its whole sub-tree is grafted under previously processed element S_{i+1}. Figure 7 shows the incoming tree for router F in Figure 5.

```
Procedure: Incoming tree update at router R

S_R  : the address space associated with router R
U: a newly received SAVE update
            U = <S, ASV, a>, where ASV=<S_1, S_2, …, S_n>
iface: the incoming interface that U arrives on
subtree(X)    : a sub-tree of the incoming tree rooted at X

[Initialization] The tree has only the root representing S_R
for (i ← n; i > 0; i-- )
  if (S_i does not exist in the incoming tree)
    if ( i = n )
      graft S_i under the root
      associate S_i with iface
    else
      graft S_i under S_i+1
  else
      if ( i = n )
        if (iface ≠ the current interface associated with S_i)
          graft subtree(S_i) under the root
          change association of S_i to iface
      else
          graft subtree(S_i) directly under S_i+1 (if not yet)
  end
```

Figure 6: Incoming tree update with a given SAVE update

2.2.4 *Handling of SAVE updates*

Upon receipt of a SAVE update, in addition to updating its incoming tree and incoming table, a router also decides whether to and how to forward the update to other routers. A SAVE update may be modified before leaving the router. Figure 8 describes the handling of a SAVE update at a router.

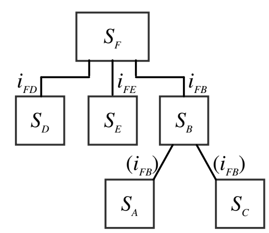


Figure 7: The incoming tree of router F in Figure 5

<u>SAVE update forwarding</u>

Whether or not to forward a SAVE update is determined by checking the SAVE update's destination address space. If a router is the last hop to reach all machines represented by the destination address space of the SAVE update, it does not forward the update. Otherwise, the next hop is determined from the local forwarding table.

To ensure that the forwarding of the update covers all routes that IP packets use to reach the update's destination address space, the forwarding table is searched for related entries. There are two types of related forwarding entries: the subset type and the superset type. A subset-type entry's destination field is a sub-area of the update's destination address space (line 7 in Figure 8); a superset-type entry's destination field is an address space covering the whole destination address space (line 21 in Figure 8). When forwarding an IP packet toward anywhere in the destination address space, the subset-type forwarding entry will be used first. If the all subset-type entries combined cannot cover the whole destination address space, the smallest superset-type forwarding entry will also be used (assuming that forwarding of IP packets uses the longest match).

Corresponding to IP packet forwarding behavior, the SAVE update is forwarded as follows. For each subset-type entry, a SAVE update is sent toward the indicated sub-area of the destination address space (lines 17 in Figure 8), and its destination address space is replaced with the sub-area address space. Furthermore, if the combination of all the first type entries does not cover the whole destination address space, the smallest superset-type forwarding entry will be used—a SAVE update is sent along the interface specified by this entry and the original destination address space is unchanged (line 22 in Figure 8). Thus a router forwards one or multiple copies of a SAVE update.

Procedure: SAVE update handling at intermediary router R.
S_R :the address space associated with router R
U : a newly received SAVE update
U = <S_D, ASV, a>, where ASV=<S_1, S_2, …, S_k> ($k \geq 1$)

1 **if** (router R is the last hop to reach all the machines in S_D)
2 **return**
3 **if** ($S_R \supseteq (S_1 \cup S_2 \cup \ldots \cup S_k)$) /* replaceable SAVE update */
4 **return**

5 **if** (a = 1)
6 ASV ←<ASV, S_R> /* append S_R; now ASV=<S_1, S_2, …, S_k, S_R> */

7 Define set E={forwarding entry e_i | e_i = <S_{Di}, out_if_i> && $S_{Di} \subset S_D$ }
8 Define an empty address space S
9 **for** every e_i in E /* inform all the sub-areas */
10 **if** (a = 1)
11 **if** (state(e_i) is *done*)
12 $a_i \leftarrow 0$
13 **else**
14 state (e_i) ← *done*
15 $a_i \leftarrow 1$
16 U_i ← <S_{Di} , ASV, a_i>
17 forward U_i along outgoing interface out_if_i
18 $S \leftarrow S \cup S_{Di}$
19 **end** loop

20 **if** ($S \neq S_D$) /* we don't entirely cover S_D with sub-areas */
21 find forwarding entry e: <$S_{D'}$, out_if'> where $S_{D'} \supseteq S_D$, such that, if there is another e_i : <S_{Di}, out_if_i> where $S_{Di} \supseteq S_D$, then $S_{D'} \subset S_{Di}$
22 **if** (e is found)
forward U along outgoing interface out_if'

Figure 8: The handling of a SAVE update

Modification of SAVE update

A router must append its own source address space to the ASV of a SAVE update whenever the appending flag a of the update is set to 1 (line 6 in Figure 8). The ASV thus records an ordered continuous sequence of address spaces crossed; such order determines the relative position of these address spaces on an incoming tree.

Overhead control of SAVE updates

If a router appends its source address space to a SAVE update, it is unnecessary to initiate another update toward the same destination. Both updates would be treated the same by downstream routers.

But it is not always necessary to append a router's source address space to a SAVE update. While a router's incoming tree should record *all* the address spaces that a SAVE update has crossed, the update's ASV is allowed to be a partial list of them, provided: (1) the rest are contained in other updates already initiated by any routers upstream; (2) combining all these updates will still provide the full sequence of the address space crossed. So, when handling a newly received SAVE update, if an intermediate router has previously initiated another SAVE update toward the same destination, the address spaces to cross after this router are already recorded in all downstream routers. The router therefore marks the new SAVE update to be no longer appendable by downstream routers (by zeroing its flag *a* as in line 11 to 12 of Figure 8).

Overhead can be further reduced by not forwarding *replaceable* SAVE updates. A SAVE update is replaceable by the router if each address space in its ASV is inside the router's source address space. The source address space in SAVE updates initiated by this router already covers the address spaces carried by the replaceable update, thus this update should be consumed by the router (lines 3 to 4 of Figure 8).

2.2.5 Conflicting SAVE Updates

If a router forwards multiple copies of a SAVE update (see Section 2.2.4 above), another router may receive several of them from different directions, but it must use only one of them to update its incoming tree regarding the common address spaces carried by these copies. In Figure 9, router R forwards two copies of the update F, one toward *r*, the other toward *R*. The latter is further forwarded from *R* to *r*. Finally, with two copies of F, *r* must decide which one to use for the area crossed prior to *A*.

When forwarding multiple copies of a SAVE update, a router calculates a priority for each copy, assigning a higher priority if the update is forwarded using a more specific forwarding entry. Router *r* in Figure 9 will thus use the higher priority update from the solid path.

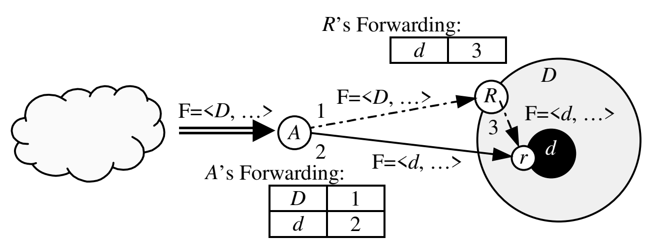


Figure 9: Conflicting SAVE updates at router *r*

3 ADVANCED ISSUES

SAVE needs to handle compatibility with legacy routers, incoming tree state maintenance, and overhead control. Mobile IP and IP multicasting also need special handling. We discuss security issues and deployment in Section 6.

3.1 Compatibility with Legacy Routers

Compatibility with legacy routers plays an important role in designing SAVE. The incoming interface information must still be correct when legacy routers are present. This requires that a SAVE update be forwarded correctly even across legacy routers. Furthermore, SAVE needs to account for address spaces of legacy routers.

Each SAVE update is carried inside an IP packet, where the destination address of the encapsulating IP header must belong to the destination address space of the update. When a legacy router receives a SAVE update, it will simply treat it as an ordinary IP packet and forward it to next hop. We are investigating the case in which multiple copies of a SAVE update are to be forwarded.

Legacy routers also complicate the maintenance of incoming trees. Since a legacy router will not send out triggered SAVE updates when its routing path to a destination is changed, the incoming tree at downstream routers will not be updated promptly. Periodic resending of SAVE updates from upstream SAVE-enabled routers solves this problem. This matches the soft-state maintenance in SAVE (Section 3.2).

Finally, if the source address space of a legacy router is not included in the source address space of a SAVE-enabled router, it is not be reported, and thus is not known to any incoming table. Thus, a SAVE-enabled router cannot easily distinguish legacy router source addresses from forged addresses. A SAVE-enabled router can discard non-existent IP addresses by checking against its forwarding table or by utilizing out-of-band information. If the IP address exists, it can switch to forwarding-table-based filtering for them. This approach will drop legitimate packets from legacy routers if the routing is asymmetric, but will properly handle many cases. We will further investigate the issue in our future work.

3.2 Soft State Maintenance

Each node's incoming tree is treated as soft state, and it can expire unless reinstated with repeated SAVE updates. Use of soft state simplifies the protocol design by automatically discarding obsolete information without the need for specific notification. When a forwarding entry is removed, it is not necessary to explicitly repair the incoming tree. Similarly it smoothly handles SAVE updates caused by transient routing behavior and asynchronous delivery of SAVE updates. Soft state also solves the problem of handling routing changes when SAVE updates cross legacy routers (see Section 3.1).

Overhead control of soft state refreshing messages is not particular to this research and has been studied elsewhere. Scalable timers [25] and a new proposal for RSVP refreshes [30] both address this problem.

Finally, soft state handles reliability issues. It has been shown that a probabilistic delivery model with relaxed reliability is suitable for soft-state-based communication, where judicious use of feedback from receivers greatly improves state consistency [20].

3.3 Overhead Control With Two-Level Routing Infrastructure

The SAVE protocol has three types of overhead: bandwidth cost, processing overhead, and storage. The number of SAVE updates sent by a router is proportional to the size of its forwarding table. Controlling SAVE update overhead was discussed in Section 2.2.4. The design there matches 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 the border router, those SAVE updates from within an AS are all *replaceable* and will not leak to the outside through this border router. In the other direction, the SAVE updates from outside an AS need to be distributed into the AS. We are investigating the aggregation of these updates.

3.4 Special Case Handling

3.4.1 Mobile IP

Mobile IP often relies on maintaining the home address of a given mobile host regardless of its location [18]. A packet from a mobile host will always carry its home IP address. With source address filtering enforced, however, such packets would be rejected whenever the mobile host is outside of its home network, since generally they use different path to the destination from the remainder of that home network.

Reverse tunneling for Mobile IPv4 has been proposed [16], by which a packet from a mobile host in a foreign network will be tunneled back to its home agent first, which then forwards the packet to the destination. In IPv6, a packet from a mobile host in a foreign network will be stamped with a care-of address, an address belonging to the foreign network. Both approaches resolve the potential conflicts between address filtering and mobile IP.

3.4.2 IP Multicast routing

IP multicast can benefit from the SAVE protocol. Multicast routing protocols such as DVMRP [8], PIM [9], or CBT [2] use a reverse-path-forwarding technique to build a *reverse* shortest-path tree. When building a multicasting tree, a SAVE-enabled router can take advantage of having an incoming table. It can determine the previous hop of the truly shortest path from the root to itself, not the reverse shortest path using RPF.

Upon receipt of a multicast packet, a router can be in one of the two phases regarding the packet: the packet is being sent towards the root where it will be further propagated towards the whole multicast group, or the packet is being propagated from the root. In the former situation, the router should validate whether the multicast packet is from the sender. In the latter situation, the source address of the root should be validated, the same way as it does for unicast packets. In DVMRP, the root is the sender; in CBT, the root is the core router of the multicast group; in PIM, it is the rendezvous point of a multicast group when a shared tree is used, and the sender otherwise.

4 SIMULATION

4.1 Simulation Goals

The SAVE protocol has been implemented and tested in a simulation environment. We have performed extensive simulation runs to obtain the following information: (1) whether all bad packets (i.e. packets with forged source addresses) can be successfully detected and dropped; (2) whether good packets (i.e. packets with authentic source addresses) are dropped erroneously; and (3) the cost of the SAVE protocol.

4.2 Simulation Design

In our simulation we assume that all routers run the incoming-table-based filtering. Corresponding to the two-level routing infrastructure of the Internet, we simulated BGP [21] for inter-domain routing and RIP [15] for intra-domain routing. Our BGP simulation implements the following policy as recommended by Cisco [14]:

- Each router in a transit domain runs BGP. A border router in a stub domain runs both BGP and RIP.
- Stub domains are non-transit domains, whether single-homed or multi-homed.
- In each stub domain, there is one preferred exit BGP router for outgoing traffic. All outgoing traffic will use this router even if alternative routes have lower link cost to some destinations.
- In a transit domain, the router prefers the routes with the shortest AS path attribute for each destination.

We used the transit-stub model from GT-ITM software to generate domain-level connectivity and intra-domain connectivity [6]. We also used the AT&T Worldnet IP backbone topology for a transit domain in some scenarios [11]. Data traffic in our simulation is UDP and generated according to Poisson traffic models.

4.3 SIMULATION RESULTS AND ANALYSIS

4.3.1 Effectiveness Verification

To verify the effectiveness of the SAVE protocol, we set up a traffic model for data packet senders. Each sender sends out both good packets and bad packets controlled by two independent Poisson processes with different rates. If the incoming-table-based filtering is effective, we expect that the distribution pattern of filtered packets over time will match the traffic model of sending bad packets. Since the asymmetric case is common in Internet-scale routing, we also include some asymmetric routes in our simulations. Asymmetric routes could be introduced by link failure, or already exist in the initial topologies as in Figure 10.

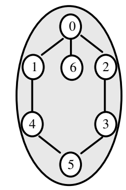
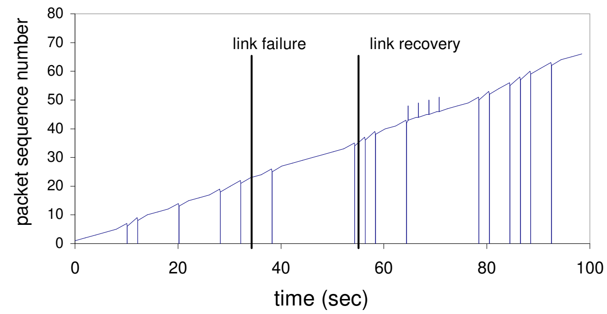


Figure 10: Topology in *Experiment 1*

The following two experiments illustrates the behavior of SAVE:

Experiment1: We evaluated the behavior of incoming-based-filtering for the topology shown in Figure 10. All packets are sent from node 6 to 5, with bad packets spoofing a source address in router 1's address space. There is an asymmetric route between node 5 and 6. When the link between 1 and 4 fails, node 0 discovers the alternate path to 5 via 2. Thus an asymmetric route is changed to a symmetric one. On recovery, routing becomes asymmetric again.

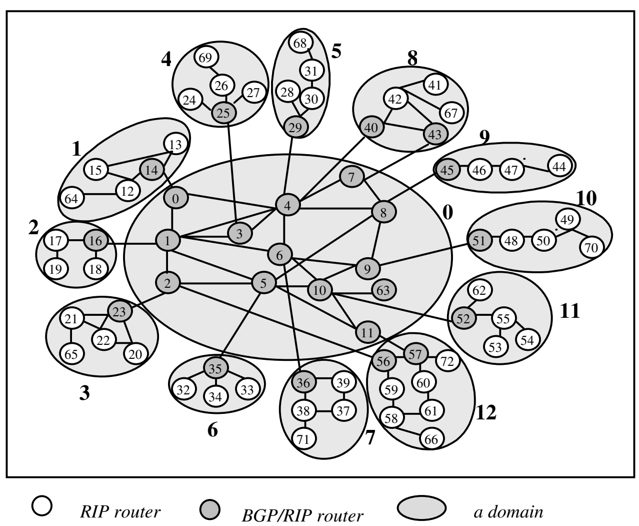


Vertical bars below the graph mark bad packets that are detected and dropped; vertical bars above the graph mark good packets that are incorrectly dropped.

Figure 11: Filtering behavior in *Experiment1*

The result is shown in Figure 11. Incoming-table-based filtering detects and drops all bad packets. Good packets that arrive on asymmetric links are not dropped while the routing is stable. When a link has failed or recovered, transient changes in routing tables result in inconsistency of incoming tables. Some good packets are dropped during that period (67 to 72 seconds).

Experiment 2: In this experiment we evaluate the behavior of the filtering mechanism for the topology shown in Figure 12 without link failures. The topology consists of one transit domain and 12 stub domains. All packets are sent from nodes 62, 66 and 70 to node 72, where bad packets spoof source addresses from the address space of router 1. An asymmetric route is formed between node 70 and 72 since 56 is the default exit router for domain 12.



A topology generated using transit-stub model, except domain 0 in the middle borrows from the AT&T Worldnet IP backbone

Figure 12: Topology in *Experiment2*

The result is shown in Figure 13. It demonstrates that incoming-table-based filtering detects and drops all bad packets, even though they are generated from multiple sources. No good packets are dropped.

4.3.2 The Cost of the SAVE Protocol

SAVE updates are analogous to routing updates in that the former are used for building incoming tables and the latter for forwarding tables. In this section, we measure the bandwidth and storage cost of the SAVE protocol and compare them to routing protocol costs. For a fair comparison, we compare RIP with SAVE using the same broadcast interval, and we compare BGP with SAVE using infinite refreshing time. Theoretical analysis of these relative costs is given in Appendix A and B.

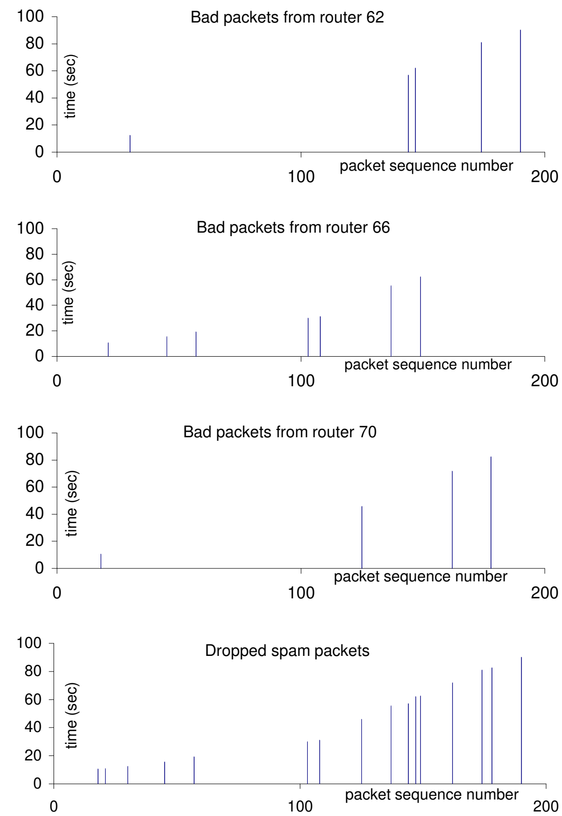


Figure 13: Filtering behavior in *Experiment2*
Every packet has a global unique sequence number. (Here, good packets are not shown.)

We have measured the following costs of both SAVE and routing protocols (RIP and BGP): (1) bandwidth—the total overhead exchanged by protocol messages in the whole network. The overhead of each message is counted every time it crosses a link; and (2) storage—the average cost per node for the incoming table and tree in SAVE and the routing table in routing protocols.

The total bandwidth cost of the protocol has two parts: (1) static overhead for initial setup; and (2) dynamic overhead for updating the incoming table and tree in SAVE and the routing table in routing protocols.

Figure 14 shows the static bandwidth cost in single-domain topologies for SAVE and RIP. Figure 15 shows the static bandwidth cost for multiple-domain topologies, where we measured the inter-domain bandwidth cost of SAVE and compared it with that of BGP. Ten different topologies were tested for each topology size.

Both graphs show that the cost of the SAVE protocol is a power function of the number of nodes, proportional to Num_Nodes^k where $2<k<3$. Bandwidth cost within a single domain is comparable to the RIP cost, while the cost of running the SAVE protocol in multiple-domain networks is approximately three times greater than BGP cost.

To measure dynamic overhead, we introduced link failures in the same topologies used to measure the static cost. We observed that the incurred bandwidth cost by SAVE varies depending on the topology and location of failed links. In some scenarios SAVE has lower cost than BGP, while in others it is at most three times larger.

Figure 16 and Figure 17 show the size of the incoming table and tree for single-domain and multiple-domain topologies, respectively. Figure 16 shows the cost of SAVE incurred for storing intra-domain information. Figure

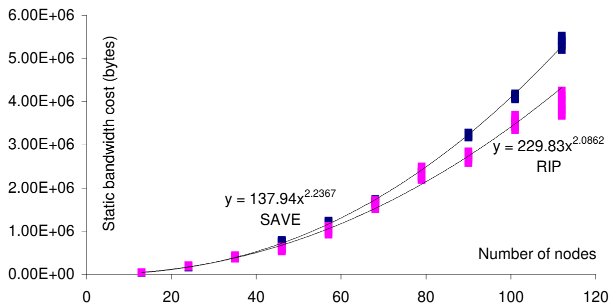


Figure 14: Static bandwidth cost comparison between SAVE and RIP

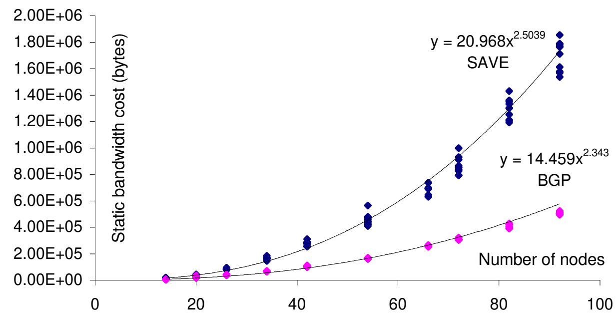


Figure 15: Static bandwidth cost comparison between SAVE and BGP

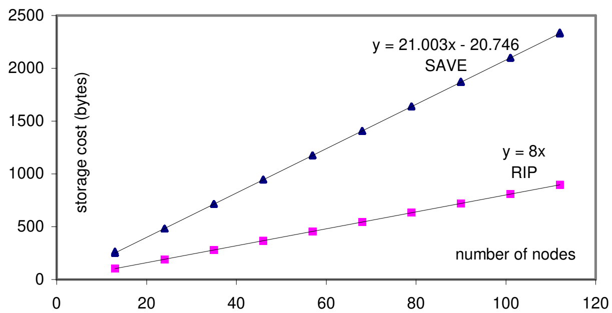


Figure 16: Storage cost for single-domain topologies

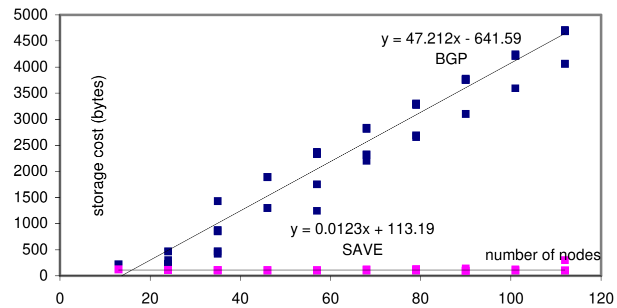


Figure 17: Storage cost for multiple-domain topologies

17 shows only the cost incurred for storing domain-level information. These costs are compared with the size of the routing tables of RIP and BGP. The storage cost of SAVE for single-domain topologies is at most three times larger than RIP's. The storage cost of SAVE for multiple-domain topologies is significantly lower than BGP's.

5 RELATED WORK

Research on network security has focused on end-to-end approaches, typically through authentication and encryption (IPsec is one representative at the IP layer [13]). To guarantee a packet's authenticity, it can be signed or encrypted. The high computation overhead of cryptographic operations prevents such approaches from being widely employed per packet. These operations also require key establishment for every pair of communicating nodes on the Internet. Finally, this approach still cannot prevent a site from being flooded by DDoS-style attacks.

IP spoofing has been addressed in other research through both preventive approaches and reactive approaches. Filtering is a preventive approach. Tracing is mostly reactive.

Filtering as a general approach has been proposed in [1], where many fields, including but not limited to source address, can be used for filtering. Martian address filtering is required to discard packets if their source addresses are special addresses (loopback address, broadcast address, etc.) or are not unicast addresses. For validation of source address in general, the forwarding table is used to validate the incoming interface of packets. This feature is often disabled by default, because it leads to erroneous packet dropping when asymmetric paths are used.

A popular filtering approach is periphery filtering that can be deployed on a firewall as well as an edge router [12]. But unless it is deployed everywhere, an attacker can still easily forge source addresses, as shown in Figure 1.

Packet tracing has been widely studied. The various approaches have their own strengths and weaknesses. In Bellovin's approach, each router samples packets with low probability and sends the sampled router adjacency information to the destination via ICMP traceback messages, allowing the destination to reconstruct the path [3]. Probabilistic packet marking encodes the path information in the ID field of an IP header [23]. This approach is incompatible with IPsec, which disallows modification of the ID field. Logging and link testing can also be used for tracing [5][26]. As pointed out in the introduction section, tracing is either expensive or ineffective. On the other hand, our filtering approach and tracing techniques are complementary. Source address filtering eases tracing, while tracing is also necessary when only partial deployment of filtering is possible.

Network intrusion detection has also studied how to localize an attacker. For instance, DECIDUOUS dynamically builds IPsec security associations to reveal the location of attacking sources [7]. However, to do this a victim running DECIDUOUS must detect the intrusion first; network topology information is also required.

6 FUTURE WORK

Open issues for the SAVE protocol include its security, aggregating SAVE updates, incremental deployment of new filtering-based routers, and incorporating this filtering mechanism with other networking techniques.

The SAVE protocol itself must be secured or attackers will merely compromise it first before taking other steps to perform their attacks. In particular, the process of building the incoming tree at each router must be protected. SAVE updates must be protected while crossing a chain of routers. End-to-end encryption provides secrecy and integrity of SAVE updates, but it inhibits intermediary routers from accessing, modifying, and using transient SAVE updates. An alternative is to use a series of signatures—digitally signing a SAVE update and re-signing its subsequent versions—to allow a destination router to verify a SAVE update's authenticity, as has been suggested by active network researchers for their own purposes [29]. Any authentication-based approach must address the fact that there is no ubiquitous authentication mechanism for the whole Internet. Since a SAVE update may traverse different autonomous domains, something must be done to provide inter-domain authentication.

Overhead control deserves further study. The aggregation of SAVE updates arriving at a router is one potential solution. The address space vector in a SAVE update, for instance, may be further aggregated. The storage of the incoming tree and incoming table can also be saved with address space aggregation.

Incremental deployment of SAVE is another open issue. One interesting problem is how to assess the benefit with partial deployment. For instance, in contrast to random deployment, if all backbone routers employ filtering, the efficacy appears more promising.

A new network protocol will be more successful if it can be smoothly incorporated with other networking techniques. We have shown SAVE's compatibility with mobile IP and IP multicasting, but there are still other arenas to consider. For instance, IP tunneling complicates source address validation in two ways. First, the true source address of a packet is buried inside a wrapping IP header that contains the source address of the ingress of a tunnel. Source address filtering could verify that the IP address of the ingress is legitimate, but could not generally determine if the true internal source address was legitimate. Second, legitimate packets that emerge from a tunnel may be dropped due to deviation from a normal path caused by tunneling. IP source routing is similar to IP tunneling in that a packet may also reach its destination via a different path than normal [10] [19]. SAVE also seeks to work with new developments in packet routing research, such as the per-hop behavior in differentiated services [4], multipath routing [27], and multi-protocol label switching (MPLS) [22].

7 CONCLUSION

Network attacks pose an increasing danger to the Internet community. The source addresses of malicious packets are often forged to hinder discovery of the attacker. Existing methods of overcoming this problem (periphery filtering, filtering based on forwarding tables, or various tracing techniques that discover the physical path of malicious packets) all have limitations in their cost or effectiveness. In this paper we present a practical and effective approach to detect improperly addressed packets.

We developed the SAVE protocol to enable routers to check source address validity. This protocol handles cases of asymmetric routing correctly. We demonstrated through simulation that the incoming table built by the protocol properly detects forged IP addresses, except during transient periods following routing changes. The incoming table produced by the SAVE protocol can be used for purposes other than filtering. Valid incoming interface information is also beneficial for many techniques (such as RPF) that currently assume symmetric routing and forward packets on non-optimal routes.

The SAVE protocol's operation is independent of the underlying routing protocol. Simulation results show that the bandwidth cost of the SAVE protocol is comparable with that of routing protocols, and its storage cost is quite small. Known optimizations could reduce this overhead in the future.

We have addressed many difficult issues for this kind of protocol, such as handling mobile IP, reliability, and some aspects of aggregation and scaling. Other hard issues will be addressed in future work, including partial deployment, security of the protocol, and more aggressive aggregation to provide better scaling properties.

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] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. "An Architecture for Differentiated Services," RFC 2475, December 1998.

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

[6] K. L. Calvert, M. B. Doar, and E. W. Zegura. "Modeling Internet Topology." *IEEE Communications Magazine 35, 6 June 1997. 160-163*.

[7] 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.

[8] 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.

[9] 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.

[10] S. Deering and R. Hinden. "Internet Protocol, Version 6 (IPv6) Specification," RFC 1883, December 1995.

[11] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan and J. E. van der Merive. "A flexible model for resource management in virtual private networks," *Proceedings of SIGCOMM' 99*, pp. 95-108, September 1999.

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

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

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

[18] C. Perkins. "IP Mobility Support," RFC 2002, October 1996.

[19] J. Postel. "Internet Protocol," RFC 791, September 1981.

[20] S. Raman and S. McCanne. "A Model, Analysis, and Protocol Framework for Soft State-based Communication," *Proceedings of ACM Sigcomm, 1999.*

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

[22] E. C. Rosen, A. Viswanathan, and R. Callon. "Multiprotocol Label Switching Architecture," Internet Draft: draft-ietf-mpls-arch-07.txt, July 2000.

[23] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. "Practical Network Support for IP Traceback," *Proceedings of ACM SIGCOMM 2000*, August, 1988.

[24] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. "Analysis of a denial of service attack on TCP," *Proceedings of IEEE Symposium on Security and Privacy*, 1997.

[25] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. "Scalable timers for soft state protocols," *Proc. IEEE Infocom 1997*.

[26] R. Stone. "CenterTrack: An IP Overlay Network for Tracking DoS Floods," *9th USENIX Security Symposium*, August 2000.

[27] D. Thaler and C. Hopps. "Multipath Issues in Unicast and Multicast Next-Hop Selection," RFC 2991, November 2000.
[28] P. Traina. "BGP-4 Protocol Analysis," RFC 1774, March 1995.
[29] V. Varadharajan, R. Shankaran, and M. Hitchens, "Active networks and security," *Proceedings 22nd National Information Systems Security Conference*, Vol.1, Arlington, VA, October 1999.
[30] L. Wang, A. Terzis, and L. Zhang. "A New Proposal for RSVP Refreshes," *Proceedings of the 7th International Conference on Network Protocols (ICNP'99),* October 1999.
[31] 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 2000.
[32] 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 A: COST COMPARISON BETWEEN SAVE AND RIP

Consider a network of N routers where every router has k neighbors. Let ES_{RIP} be the entry size of a routing table. Every RIP router periodically broadcasts updates to all k neighbors. The size of every update is proportional to N. Total bandwidth cost per broadcast per router is $BWC_{Broadcast}^{RIP} = k * N * ES_{RIP}$. We assume that for every topology change there is a probability p_e that it will affect a given entry in a routing table. The change of this routing entry triggers updates to all k neighbors. Total bandwidth consumed by all triggered updates per topology change is $BWC_{Trigger}^{RIP} = p_e * k * N * ES_{RIP}$. The storage cost per router is $SC^{RIP} = N * ES_{RIP}$.

In SAVE, each router generates a SAVE update for every forwarding table entry and sends it towards the destination. Let U_A denote the size of address space information within a SAVE update, and d be the mean diameter of the network. Every SAVE update is forwarded d hops on the average, and every router on the path appends its address space to the update. Total bandwidth consumed per router is $BWC_{Broadcast}^{SAVE} = \frac{U_A * d * (d+3) * N}{2}$. We assume that every routing table change affects the corresponding entry in the forwarding table and triggers SAVE updates. The total incurred bandwidth per router is thus $BWC_{Trigger}^{SAVE} = \frac{p_e * U_A * d * (d+3) * N}{2}$. Let ES_{SAVE} and ES_{TREE} denote the size of an entry in an incoming table and tree. The storage cost per router is $SC^{SAVE} = N * (ES_{SAVE} + ES_{TREE})$. The ratio of bandwidth cost of the SAVE protocol vs. bandwidth cost of RIP for both broadcast and triggered updates is $RBWC = \frac{d * (d+3) * U_A}{2 * k * ES_{RIP}} = O(\frac{d^2}{k})$. The storage cost ratio of SAVE vs. RIP is $RSC = \frac{(ES_{SAVE} + ES_{TREE})}{ES_{RIP}}$.

APPENDIX B: COST COMPARISON BETWEEN SAVE AND BGP

Let A denote the total number of ASs on the Internet, M the mean AS distance (in terms of the number of ASs), and N_W the total number of networks. We assume that the networks are uniformly distributed among the ASs, and every BGP router peers with k other BGP routers.

For BGP cost evaluation, we use the discussion in [28]. After the initial BGP connection setup, the peers exchange a complete set of routing information, and each BGP update groups N_W/A NLRI entries for every AS. Each BGP router sends routing information about all N_W networks. Denote ES_{NLRI} the size of a NLRI entry in a BGP routing table, and ES_{AS} the size of the AS-PATH attribute. The average size of a BGP update is thus $BWC^{BGP} = (N_W / A) * ES_{NLRI} + M * ES_{AS}$. The complete routing information consists of A such updates. The total bandwidth used during the setup phase per router is $BWC^{BGP}_{Setup} = k * (N_W * ES_{NLRI} + M * A * ES_{AS})$. We assume that for every topology change there is a probability p_e that it will affect a given entry in a routing table. The average bandwidth cost per topology change for each router is $BWC^{BGP}_{Change} = p_e * N_W * k * (ES_{NLRI} + M * ES_{AS})$. The storage cost for each BGP router is $SC^{BGP} = k * N_W * (ES_{NLRI} + M * ES_{AS})$.

We compare the bandwidth cost of inter-domain SAVE updates with the cost of BGP updates. Let D denote the mean inter-domain distance in terms of the number of hops. At initialization, every border router sends a *SAVE* update to each destination network in its forwarding table. Let U_A denote the size of address space information in a SAVE update. Every update, on the average, travels D hops before it reaches the destination network[2], and every router on the path appends its address space information to the update. The total bandwidth used by a router for the initial setup is therefore $BWC^{SAVE}_{Setup} = \frac{N_W * U_A * D * (D+3)}{2}$. If every routing change leads to a change in the forwarding table, the bandwidth of trigged *SAVE* updates per topology change is $BWC^{SAVE}_{Change} = \frac{p_e * N_W * D * (D+3) * U_A}{2}$. Since a SAVE router stores the incoming table and tree at the AS level, the storage cost of SAVE is $SC^{SAVE} = A * (ES_{SAVE} + ES_{TREE})$. The bandwidth cost during setup of SAVE relative to BGP is $RCBW_{Setup} = \frac{N_W * U_A * D * (D+3)}{2 * k * (N_W * ES_{NLRI} + M * A * ES_{AS})} = O(\frac{N_W * D^2}{k * (N_W + M * A)})$. The bandwidth cost in the change phase of SAVE relative to BGP is $RCBW_{Change} = \frac{N_W * D * (D+3) * U_A}{N_W * k * (ES_{NLRI} + M * ES_{AS})} = O(\frac{D^2}{k * M})$. The storage cost of SAVE relative to BGP is $RSC = \frac{A * (ES_{SAVE} + ES_{TREE})}{k * N_W * (ES_{NLRI} + M * ES_{AS})} = O(\frac{A}{k * N_W * M})$.

[2] Note that once the update reaches the destination network it still needs to be distributed to all interior routers. This incurs some bandwidth cost. Let n be the number of interior routers and d the average distance from them to the border router, then there will be additional $U_A * n * d * (d+3)/2$ bandwidth consumed for each SAVE update that reaches the border router. Since we are calculating pure inter-domain cost of SAVE, we do not include this additional cost here.