IMPLEMENTING ADDRESS ASSURANCE IN THE INTEL IXP ROUTER

Michael Burns, Gregory Prier, Jelena Mirkovic, Peter Reiher

UCLA

{mburns, greg, sunshine, reiher}@cs.ucla.edu

ABSTRACT

Many desirable network security features require new functionality in routers. Using programmable routers is an attractive approach to testing such security functionality and may serve as an easy path to deploying it. This paper describes an Intel IXP implementation of a protocol designed to help combat IP source address spoofing. It describes the problem of IP spoofing, presents a protocol that helps handle spoofing, and discusses why the Intel IXP 1200 router is an attractive platform for developing the protocol. The paper also presents the design of the IXP implementation and gives performance results that confirm the suitability of the IXP for this task.

1. INTRODUCTION

The Internet's well known security weaknesses are potential causes of costly disasters, so significant research efforts are being directed to correcting these deficiencies. Many recently proposed methods for improving Internet security require assistance from routers. In some cases, this assistance can be provided by functionality already existing in modern routers, but in other case some additional functionality must be added to support the security method. Because standard routers tend to be difficult to change, some promising security features are not yet present in the network, and research on others is impeded.

Programmable routers offer an attractive solution to this problem. They allow customizations, not just of a router's control plane, but also of its data plane. Programmable routers are a more realistic experimental platform than the software routers typically used in today's research, because they better approximate the speed and architecture of real Internet routers. Ultimately, programmable routers may offer an effective way to introduce security functionality into the Internet without requiring new router hardware upgrades for each new security feature.

This paper describes the implementation of an address assurance protocol called iSAVE on the Intel IXP 1200 programmable router. The goal of the implementation was to determine if the Intel IXP programmable router offers a useful platform for testing network security functionality at high scale. By implementing an important network security system on the Intel IXP, we were able to demonstrate that a programmable router could provide such functionality while still offering high-speed handling of packets.

The address assurance protocol we implemented was designed to combat IP spoofing, a security flaw in the IP protocol that allows attackers to forge source addresses in their packets. Attackers commonly spoof the source address in attack packets to make it harder to trace their activities, to avoid filters designed to remove their attack traffic, and generally to prevent defenders from assigning responsibility for attacks to particular machines. Until recently, the only methods available to combat IP spoofing either required very computationally expensive cryptography or were only effective when deployed ubiquitously.

The address assurance project that led to this research has worked on network protocols that will allow cheap detection of IP spoofing than the use of cryptography, with more reasonable deployment requirements. In essence, these protocols build tables that match portions of the IP source address space with interfaces at a given router. A packet with a particular IP source address must arrive on the proper interface indicated in these tables, or it is regarded as a spoofed packet. In this way, a packet identified as spoofed could be dropped or given some form of special treatment.

The SAVE protocol [Li02] works well if every router in the network deploys it, an unreasonable restriction in today's Internet. The iSAVE protocol [Mirkovic02], on the other hand, was designed to achieve similar effects in the more realistic case of partial deployment. Analysis by Park [Park01] suggests that the tables created by a protocol like iSAVE created at 18% of Internet autonomous domains would be sufficient to filter out almost all spoofable packets.

We built the iSAVE protocol in a simulation environment, but had no good evidence that it could be run at sufficiently high speed to be effective in a real router. Filtering forged packets using iSAVE tables requires that each packet's source address be checked at each iSAVE-capable router it visits. By porting iSAVE to the Intel IXP programmable router, we were able to demonstrate that a high-speed router can perform iSAVE functions without performance penalty and can forward packets as quickly as the same router without iSAVE. This result both shows the practicality of iSAVE on a high-speed router and demonstrates the value of using programmable routers to experiment with network security mechanisms.

Section 2 describes the problem of IP spoofing in somewhat more detail, discussing previous solutions to the problem. Section 3 gives a description of the iSAVE protocol, concentrating on the details most pertinent to the IXP implementation. Section 4 briefly covers details of the IXP architecture that must be understood to comprehend the iSAVE design. Section 5 describes the architecture of iSAVE on the IXP hardware, comparing it to the pure software implementation. Section 6 presents performance data that demonstrates the protocol's performance on the IXP hardware, contrasting it to the hardware performance that can achieved without including iSAVE functionality. Section 7 discusses related work, and Section 8 offers our conclusions and the lessons learned from performing this research.

2. IP SPOOFING

In the existing Internet architecture, routers typically make no attempt to determine the validity of the source field in an IP header. This field is intended for the IP address of the machine that originated the packet. Many protocols and applications address their response packets to the address specified in this field, so if it is forged, all responses will be misdelivered to the spoofed address, rather than the true one. Thus, normal users and applications have a strong incentive to use proper addresses in this field.

Attackers have different goals that make source address spoofing useful to them. When attackers send packets like the Ping of Death (which is intended to cause the target machine to crash) [Bremford] or are performing a distributed denial of service attack, delivery of the attack packets is the entire goal. Receipt of the responses is irrelevant to their purpose.

Attackers also use spoofing to avoid leaving traces that indicate where the attack originated. If they left their true address in the packet, analysis by the victim could make the identity of the attack machine known. Targets could then take any number of actions, from filtering out packets originating at the attack machine to calling law enforcement authorities to investigate the owner of the machine. These actions are difficult or fruitless if the address is spoofed. Thus, attackers often have no reason not to spoof the source address, and good reasons to do so.

Address spoofing by attackers has long been known to be a problem, but the available solutions are limited. Operating systems can enforce true packet addresses, but often their enforcement mechanisms are trivial to bypass.

Routers located at the point where a local network connects to the Internet have some power to ensure proper addresses. Such routers can be configured to know the entire range of legitimate addresses in their local network, and that range represents the entire set of legal source addresses for packets going through the router toward the Internet. This method, called ingress filtering [Ferguson00], has long been a recommended practice for installations running edge routers. Unfortunately, it has the disadvantage that those who must configure it at their routers gain little direct advantage from doing so, and may lose some desirable flexibility. For whatever reason, years of requests for network administrators to enable ingress filtering on their edge routers have gone largely unheeded.

A destination node can also verify the source of its packets cryptographically. IPSec offers this capability, as do various virtual private network protocols. These methods cryptographically authenticate the source IP address, along with most or all of the rest of the packet. With the usual assumptions related to security of cryptography, these methods ensure that the source IP address is valid. Unfortunately, they do so at high computational costs. Further, they presume that the source and destination are able to verify each other's identity in order to set up secure cryptography. Such verification requires certificates or some other heavyweight authentication system. As a result, relatively few sites use these cryptographic methods to validate source IP addresses.

Until recently, these possibilities exhausted the list of methods for defeating IP spoofing. In the last few years, however, researchers began to examine other methods based on determining validity of a packet's source IP address while it is "in flight," usually at routers in the Internet.

One simple proposal of this kind was to use the existing forwarding tables to check the validity of IP source addresses. In essence, the router would check the source address of the packet against the interface that the router would use if that address were a destination. If routing is symmetric at that router, the same path would be used to send packets to an address as to receive packets from the address, so the forwarding table entry would properly indicate the correct interface for the packet to come in on. Unfortunately, a large proportion of the traffic in the Internet is asymmetric. This simple scheme would improperly filter all asymmetric traffic, which would be disastrous. The existing forwarding tables are essentially only one-way and do not contain information indicating such asymmetries, so a simple solution based on current forwarding table contents will not work.

SAVE [Li02] and iSAVE [Mirkovic02] are two more sophisticated protocols that belong to this new family of solutions. While these protocols differ in many details, their goal is the same: to construct router tables that map ranges of IP addresses to the proper incoming interface for packets with those source addresses. For example, in Figure 1, packets originating at IP address a.b.c.d clearly should enter router R on interface 1, not on interface 2, so these protocols should have a table entry listing that a.b.c.d maps to interface 1. This functionality can be present at any router in the network, not just at edge routers. When a packet arrives on an interface different than that stored in the table, the router will deduce that the packet's source address was spoofed. The packet can be dropped, flagged as suspect, or treated in some other special manner. Analysis by Park [Park01] suggests that if a relatively modest (but well-chosen) 18% of all of the autonomous domains in the Internet had routers that hosted such tables, practically all spoofed IP packets would be detected by them.

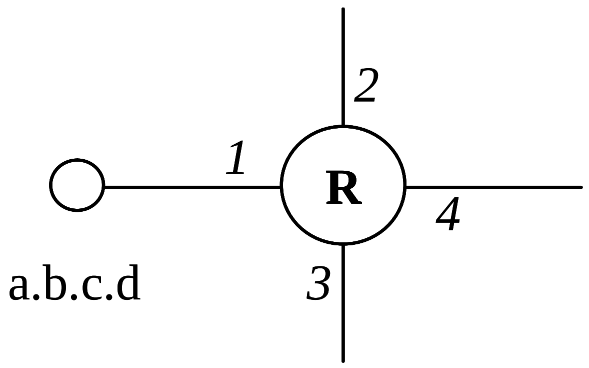


Figure 1. A router with several interfaces

While SAVE and iSAVE share similar goals, the protocols are actually quite different. iSAVE is more suited for practical deployment than SAVE, since it can still operate correctly when partially deployed. Thus, while we could have implemented SAVE on the Intel IXP routers, we chose to implement iSAVE.

3. THE ISAVE PROTOCOL

The iSAVE protocol provides participating routers with the information needed to detect a packet with a spoofed source IP address, based on the incoming interface on which it arrives. iSAVE routers exchange *iSAVE updates* specifying the legitimate incoming interface for a range of source addresses. This information is then stored in a data structure called *incoming table* and used for spoofed packet filtering. The incoming interface of each data packet is compared against the expected interface from the incoming table entry specified by the source address in packet's IP header. Matching packets are forwarded further and mismatching packets are dropped.

The iSAVE protocol contains a large number of details that are vital to correct operation of the protocol, but are not of special relevance to the port of iSAVE to the IXP programmable router. The description below includes all details required to understand the IXP implementation. A more complete description of iSAVE can be found in [Mirkovic02].

The goal of the iSAVE protocol is to build incoming tables at participating routers quickly, efficiently, and correctly, and to specify how they can be used for spoofed packet filtering. The iSAVE update generation and filtering functionality represent two major parts of iSAVE and are described separately, although they can both be deployed at a single router. *Informational routers (I-routers)* generate iSAVE updates to establish legitimate incoming interface information for some range of source addresses along a set of routes. *Filtering routers (F-routers)* on these routes intercept the updates and use them to build an incoming table and perform packet filtering.

An I-router is assigned responsibility for a *source address space (SAS)* — a set of IP addresses that use this router as an exit router to reach the Internet. This source address space can be configured through administrative means or through a separate protocol. The packets from a given source address space *A* can reach numerous destinations. However, Internet communication patterns are such that at any given time, only a small subset of these destinations, *B*, receives either legitimate or spoofed traffic carrying source addresses from *A*. Consequently, only the filtering routers along routes from *A* to *B* must have correct incoming table entries to perform filtering. This observation helps an I-router minimize the overhead for iSAVE update generation while still providing correct information for packet filtering to relevant F-routers.

An I-router detects the set of currently active destinations by monitoring the destination IP addresses of data packets it relays for its source address space. It then sends *proactive iSAVE updates* along routes to these destinations. An iSAVE update contains the source address space of the issuing I-router and is addressed to a single destination, thus ensuring that the validated route is the same one that data packets would take to reach the given destination.

To limit the costs of updates to an I-router, only a subset of active destinations is chosen for route validation; the amount of free resources available on an I-router's determines the size of the subset.

Other factors further limit the number of updates that must be sent by I-routers, lowering the cost of the protocol. Routes to many destinations in the Internet overlap, and a single iSAVE update can validate a significant portion of a route to a set of destinations. To take advantage of that, an I-router generates updates destined for 24-bit IP prefixes, rather than to single destinations, allowing a single update to validate the route for the entire set of destinations sharing the prefix. This aggregation may not always be correct, but according to figures derived from a study of the BGP routing system [Oregon 02], out of 118,164 routing prefixes, only 2328 prefixes have a mask length longer than 24. This means that destinations sharing a 24-bit prefix follow the same route with a probability of over 98.03%.

The I-router also limits its costs by maintaining a hash table of the destination prefixes for which the update has been sent and associates a timer with each entry. A new update will not be sent to the same prefix until a timer has expired.

Although proactive updates validate the majority of routes, they are not sufficient to build all the incoming table entries required at relevant F-routers for a number of reasons:

- Since an I-router does not validate routes to all currently relevant destinations due to the large overhead of such an approach, some F-routers that see legitimate traffic from the protected SAS may not have proper incoming table entries for that SAS.
- Proactive updates only validate routes to destinations of legitimate traffic, creating incoming table entries at F-routers along those routes. Spoofed traffic could go through F-routers that never see legitimate traffic from the protected SAS, and thus would not have a corresponding incoming table entry to perform filtering.
- Routing changes can alter routes and thus effectively make existing incoming table entries incorrect. Between the time a routing change occurs and the I-router next chooses to send a proactive update to the same destination, the invalid entries will persist and will cause improper handling of both spoofed and legitimate traffic.

iSAVE handles these situations through an *on-demand* update mechanism. If an F-router receives a data packet for which it has no incoming table entry (or, in some cases, has an incorrect entry), it will issue an *iSAVE request* message. The data packet is then forwarded to the destination, as are other similarly addressed packets that arrive while waiting for the response. This decision may cause some spoofed packets to be delivered to a destination, but it ensures that proper packets are not dropped simply because the reply message has not been received, and ensures that the system does not drop proper packets from an address space where iSAVE is not deployed. An iSAVE request contains the destination IP address of the data packet (*dIP*) and is sent toward the alleged source of the data packet (*sIP*). On the route to *sIP*, the iSAVE request will be intercepted by either the authoritative I-router for the source address space encompassing *sIP* or by a border router that recognizes iSAVE requests and redirects them to the appropriate I-router. The I-router then generates a *reactive iSAVE update* and sends it toward *dIP*. This update establishes correct incoming interface information at all F-routers along the route, including the F-router that actually made the request.

Requesting updates when no incoming table entry exists does not perfectly handle all situations. For example, if a routing change causes packets for an SAS to enter an F-router on a different interface than its current incoming table entry indicates, the F-router will improperly filter them. Or if a routing change causes packets from a particular source to no longer flow through an F-router that has an incoming table entry for that source, the F-router will never see another proactive update, but will still have an entry for the SAS that is no longer correct. If nothing else were done, that entry would allow packets with spoofed addresses from the SAS to flow through the F-router. These and other similar situations are handled by associating a timer called T_{fresh} with each entry in an F-router's incoming table. T_{fresh} is refreshed every time an iSAVE update for a given entry is received. When T_{fresh} expires, the entry is no longer used for filtering. Instead, if a mismatching packet arrives, the F-router will send an iSAVE request to the packet's source address. Generating an iSAVE request for each mismatched packet seen by an F-router would open the possibility for a denial-of-service attack on the router; a large stream of spoofed packets with different source addresses would lead the router to generate a request for each, soon depleting its resources. Many of these requests would also be redundant since only the first one, if not lost, suffices to invoke the desired iSAVE update generation. An F-router thus spaces out its requests for a given source address space. It associates a timer $T_{pending}$ with each entry for which a request has been sent. A positive value of this timer prevents sending further requests for this entry if mismatching packets arrive. Instead, a default forward policy is applied to mismatching data packets while waiting for the iSAVE update. The initial value of the $T_{pending}$ timer is set to be several times larger than the estimated round-trip time to the given source address space.

If the desired update arrives before the $T_{pending}$ timer expires, the timer is removed, the entry is updated and T_{fresh} is reset. Otherwise, if $T_{pending}$ expires, one of three events has occurred:

- The iSAVE request has been lost.
- The iSAVE update, replying to the request, has been lost.
- The source address space is a legacy space (or an iSAVE protected space whose I-router cannot currently satisfy the request).

To account for the cases of lost requests and responses, the F-router will retransmit the request and restart $T_{pending}$ timer with an exponentially larger value. After N_{trial} unsuccessful requests, the source address space is assumed to be under control of a legacy router, rather than an I-router. In this case, the entry will be removed from the incoming table.

In some cases, the path between the attacker generating the spoofed packets and the attack’s destination crosses an F-router that is not on the path from the misused source address to that destination. iSAVE requests generated from this router will ask for an update to be delivered to the destination, thus the F-router would not see a response for any of these requests. To handle this case, after the first few requests have gone unanswered, the F-router asks that the response be delivered directly to itself, rather than the destination address, and thus obtains the information necessary to filter out spoofed packets.

Legacy address spaces should not be bothered by frequent iSAVE requests, since they cannot respond to them. An F-router achieves this effect by storing all spaces that are determined to be legacy in the *legacy cache*. Data packets arriving from these spaces are forwarded without any checking of incoming interface information. Legacy address spaces can become iSAVE protected spaces. To account for this case, the F-router does not keep entries in the legacy cache forever, but associates a T_{legacy} timer with the newly inserted entry, and deletes the entry when the timer expires. If an iSAVE update is received from the legacy address space, the entry will be deleted and inserted in the incoming table.

The purpose of the incoming table is to store information for currently active source address spaces, and not for all known source address spaces. This approach limits the memory consumption for incoming table entries. An entry is considered active if data packets or iSAVE updates for it have been received recently. An F-router keeps a T_{active} timer associated with each entry, and resets it whenever an iSAVE update or data packet for that SAS (matching the specified incoming interface or not) is received. If T_{active} expires, the entry is deleted from the incoming table.

The iSAVE protocol was tested in a simulation form, using Javasim. The IXP implementation is the first actual implementation of iSAVE.

Appendix A shows a flowchart that describes the behavior of an iSAVE F-router.

4. THE INTEL IXP ARCHITECTURE

The following section covers only the general design of the IXP 1200 and details that are important to the iSAVE implementation. For more specific information, please refer to [Johnson02] and [Intel].

The Intel IXP programmable routers, like most routers on the market today, offer two different paths for packets. Six microengines allow fast path packet handling and a StrongARM core processor supports slow path operations. The microengines are designed specifically for packet processing and run special code written solely for them. Developing this code is like programming an embedded system. The microengines have a limited program store, but access to larger amounts of memory shared with the StrongARM processor. The StrongARM processor forms the slow path as well as the control plane of the router. It should handle infrequent exception events and responses that are too complex for processing in the streamlined, packet-driven, fast path.

Microengines have access to several forms of memory. There is 4K of very fast scratch memory, 8 Mbytes of SRAM, and 256 Mbytes of SDRAM. This memory is shared among the microengines. Each memory type has different memory access restrictions. SRAM and scratch memory accesses must be word aligned and SDRAM must be double word (8 byte) aligned [Johnson02]. The microengine hardware also provides some commonly used functions and data structures in memory, such as hash functions and stacks. These allow hardware parallelism and are faster then the corresponding software implementations.

The microengines start as bare processors with no functionality. Everything must be built from the ground up. The application must be broken into distinct chunks, and then the fast-path parts of these chunks must be written and assigned to microengines. The slow-path parts must be written as well and run on the StrongARM. Luckily, Intel provides some sample applications such as IPv4 packet forwarding and layer 2 bridging so that developers can start with code implementing basic router functionality.

The slow path is implemented in the on-board StrongARM core processor. This processor runs a version of the Linux operating system. It shares both SRAM and SDRAM memory with the microengines. The application runs just like any other program, but has special hooks to access the shared memory area. Synchronization primitives are provided to give mutual exclusion to either the StrongARM or a given microengine.

5. THE IXP IMPLEMENTATION OF ISAVE

Our current IXP iSAVE implementation includes only the F-router functionality. The vast majority of the I-router functionality occurs in the router's control plane, not in the data plane, while critical components of the F-router functionality (such as filtering and entry updating) must be performed in the data plane. Therefore, we decided that we would learn more about implementing security protocols on the IXP by tackling the F-router first. This implementation assumes that there are suitable I-routers performing their part of the protocol. The remainder of this section will only discuss the F-router implementation.

The F-router functionality is divided between data plane processing and control plane processing. Generally, as much functionality as possible was moved out of the data plane. This was one of the driving goals of the design, leading to many differences from the previous implementation of iSAVE. While both microengine-based data plane and StrongARM-based control plane functionality have been implemented and tested, this is still classified as a work-in-progress with issues remaining in optimization, scalability, and security.

5.1. The Microengine (Fast Path) Component

The microengine component examines all incoming packets to determine source address validity. A single microengine with four hardware-supported threads receives packets on the fast path from each of four 100-Mbit ethernet interfaces. When the microengine finishes receiving a packet, it looks up its source address in a trie stored in the SRAM. This data structure is derived from the trie data structure used by Intel's sample IPv4 packet-forwarding code to perform route lookups using a packet's destination address. While the IPv4 code expects to always have some default entry for any possible destination address, iSAVE's data trie might well have no entry at all for a particular source address. Furthermore, iSAVE domain entries are never subsets of another entry, meaning that the use of interior nodes is not necessary. The iSAVE trie implements the incoming table, the legacy cache, and the pending cache in a single data structure.

The trie is indexed by four-bit segments of the source address. At each level in the trie, there are 16 32-bit word entries. If the F-router has no information about proper interfaces for any addresses represented by a particular node of the trie, that node will be an empty leaf node. If there is one proper interface for all addresses represented by the node, the node is a leaf node containing that information. If there is information for a part of the space, or different information for different parts of the space, the node points to a lower-level set of 16 words that divide up the next four bits of the source address.

If no entry exists for an address range being looked up, a new node needs to be added to the trie. This operation is not conducive to fast-path execution because of expensive memory allocation operations and complex data structure manipulations. Therefore, in this case the fast-path iSAVE code flags the packet with a NO_ENTRY exception and sends the packet up to the StrongARM core. The slow-path processing will then handle the addition and initialization of a new entry in the trie structure.

If an entry exists, the microengine sets its *active bit*. The active bit is a single bit reserved in a non-interior trie entry for the purpose of determining whether traffic with a source address in the range of an entry has passed through this router recently. Periodically, the StrongARM component examines the active bits of all trie entries, removing those entries whose bits are not set and clearing those bits that are. This process prevents an entry from remaining in the trie permanently after its associated address space no longer sends traffic, which may occur in the event of a routing change.

The entry may or may not be considered valid. Valid entries are those that are still fresh, as described in Section 3. Information that an F-router obtains from an I-router about the proper interface for a particular range of IP source addresses is only regarded as authoritative for a limited period of time. After that time, the entry is saved for later use, but is not trusted for

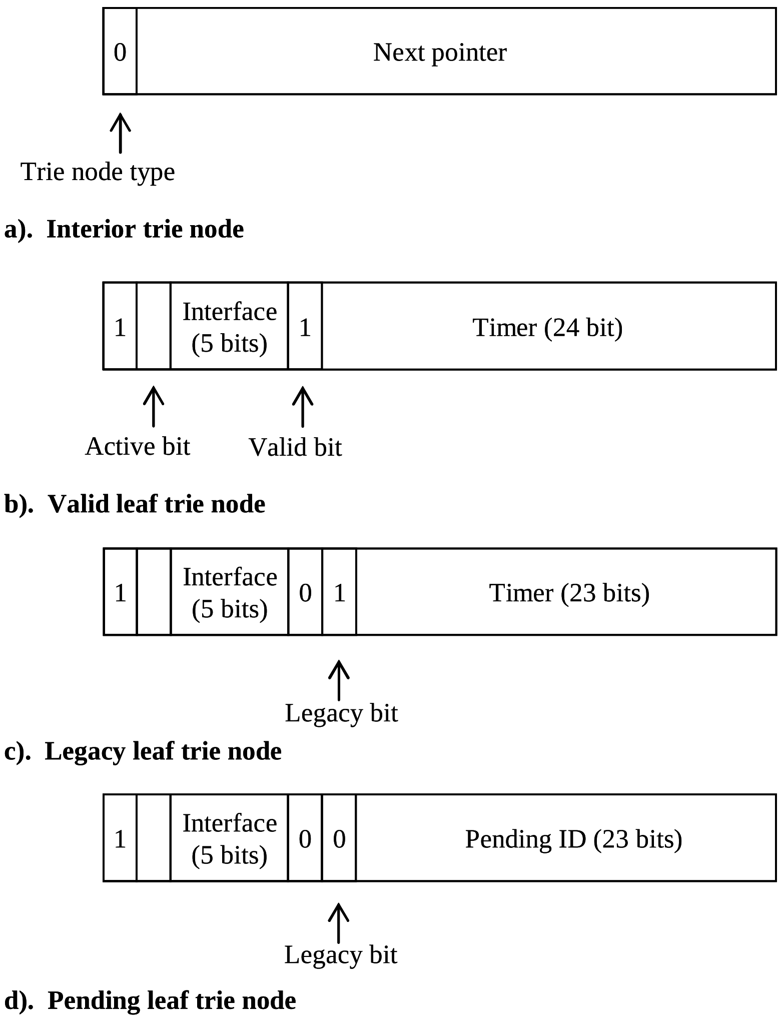


Figure 2. The Trie data structure

filtering spoofed packets. Another bit in each trie data entry is reserved to mark an entry as valid. If the entry is valid, the packet's interface is compared to the interface listed in the trie entry. If they match, the packet moves on to IPv4 route lookup and is eventually passed to the microengine reserved for transmitting packets.

The valid bit in the entry is manipulated in a lazy fashion. For valid entries, 24 bits of the entry are used to store a timestamp. This timestamp is checked when the listed interface mismatches the actual incoming interface of the packet. The timestamp represents the latest time we are allowed to enforce this entry, and is some small number of seconds from when the entry is updated by the core. If this timer has expired, the entry is converted from a valid entry to a pending entry, indicating that the router is expecting to hear an update from the source address I-router. The microengine itself does not create the iSAVE request message to send to that I-router. Rather, it marks the packet with a CHECK_ENTRY exception and passes it to the StrongARM for remaining processing. If the entry is valid, the timer has not expired, and the interface does not match, the packet is dropped in the belief that its source address is forged.

If the entry is not valid, it is either a pending entry (indicating that the router is in the process of asking for or receiving updated information on the source address) or a legacy entry (indicating that the router believes there is no I-router in charge of the source address). If the entry is pending, as far as the microengine is concerned, the StrongARM has already been alerted to the need for an update, so nothing more needs to be done. The packet is simply passed to its destination. The StrongARM will handle the repeating requests to the proper I-router and will not be alerted by the microengines for this pending entry until it is no longer pending.

In the case of a legacy entry, the situation is slightly more complicated. As with valid entries, legacy entries should timeout, since what appeared to be a legacy SAS yesterday could be an iSAVE-protected SAS today. Again, as with valid entries, the IXP implementation handles the issue lazily. There is a timestamp in legacy entries, but it is not checked unless traffic comes through the router. If traffic does come through, the timestamp is checked, and if it has expired the packet is flagged with a CHECK_LEGACY exception and passed to the StrongARM for further processing. The entry is marked pending to reduce the number of exception packets passed to the core, as with expired valid entries.

The remaining case is that the trie entry indicates that the SAS is a legacy space and the timer has not expired. This case represents the situation where we have reason to believe that the source-address space is really not iSAVE-protected. The F-router should not try to filter such packets nor pester the routers in the address space with iSAVE requests that they are not equipped to understand or respond to. Thus, for legacy entries with timestamps not yet expired, the microengine continues normal packet processing and passes the packet to be transmitted.

As the discussion above suggests, the structure of a trie node is a bit complicated. This complication arises from a desire to minimize the size of the nodes for scalability reasons. Figure 2 graphically recapitulates the data structure of the trie's nodes for different types of nodes. Figure 2a shows the structure of an internal trie node and Figure 2b shows the structure of a valid trie node. Valid trie nodes will cause filtering of packets whose interface does not match the entry's interface. Figure 2c shows a legacy trie node. These nodes represent address spaces that the F-router believes to have no associated I-router. Such nodes will not result in filtering. They are kept in the trie so that no iSAVE request will be generated when traffic from these address spaces arrives. Figure 2d shows a pending trie node. Pending nodes are awaiting a response to a request sent to the I-router handling their address space. No filtering is performed on them, and the purpose of having the node is again to avoid generating redundant slow-path exceptions.

The microengine also handles iSAVE update packets, but it merely flags update packets as UPDATE exceptions and passes the packet to the StrongARM for further processing.

5.2. The StrongARM (Slow-Path) Component

Much of the functionality of the iSAVE protocol is implemented in the IXP's StrongARM processor. Since this can be programmed in C++ for the Linux environment, most of the functionality is similar to what would be done in the control component of a software router. Generally, as described in Section 5.1, anything requiring more than trivial handling is passed from the microengine to the StrongARM. This includes handling proactive iSAVE updates, sending iSAVE requests, and dealing with associated responses. The StrongARM also handles removal of obsolete nodes from the trie and creation of new trie nodes as more detailed information becomes available.

The microengine flags packets with exceptions to signal the StrongARM that actions are required. As noted earlier, these exceptions include UPDATE (to handle an incoming proactive update or request response), NO_ENTRY (when no trie node exists for the source address of an incoming packet), CHECK_ENTRY (to determine if an existing match of source address to incoming interface is still valid), and CHECK_LEGACY (to determine if a SAS is still a legacy address space).

If the UPDATE exception indicates that the fast path has passed an iSAVE update to the StrongARM, this update may either be proactive or in response to an iSAVE request. If it is a proactive update, the StrongARM determines if a node already exists in the trie for the source-address space in the update payload. If so, it changes the indicated interface in that entry to the interface the update came in on. If not, the StrongARM creates a new trie node, sets its contents to the proper interface, and inserts it in the proper place in the trie structure. In either case, the StrongARM sets the validity timestamp in the data node to some time in the future. In some cases, an UPDATE will cause consolidation of entries and removal of nodes from the trie. Responses to requests are handled similarly, but require extra steps to clean up pending information, as discussed below.

The NO_ENTRY exception indicates that the microengine has flagged a packet with a source address that was not currently represented by a node in the trie. The StrongARM has two tasks here; it must create a node in the trie and send a request to the I-router that handles the source address in question for updated information. The StrongARM initializes a new 32-bit trie node and inserts it into the trie structure. This node is in the form indicated in Figure 2d, so the final 23 bits here represent a pending identifier. This identifier indexes an entry in a pending cache kept in the StrongARM's memory. The pending cache entry contains complete information on the pending request, including the source address in question and a timer that will expire if the reply does not come in time. On expiration, the StrongARM will send another request, with the timer reset with an exponential backoff. After sufficient failed attempts to elicit a reply, the StrongARM component will deduce that the source address is not protected by an I-router and will change the trie node to indicate that the source address is in a legacy space. If a reply eventually comes in, it is handled as an update, except that the entry in the pending cache is removed.

A CHECK_ENTRY exception is raised when an existing trie entry's valid flag is turned off. The StrongARM handles this case much as it handled the NO_ENTRY case, except that it need not create a new trie entry. It merely sends a request and creates an entry in the pending cache.

Currently, a CHECK_LEGACY exception is handled as a CHECK_ENTRY exception. This may change in the future to accommodate trie entry aggregation and the necessary additional actions to be performed when an aggregated legacy space needs to be probed. This method of handling the CHECK_LEGACY exception differs from the iSAVE protocol described earlier, and represents an experimental optimization.

The StrongARM also has responsibility for periodically scanning the trie for inactive nodes. This process is currently done once per day, but can be done at different frequencies, depending on the congestion of the trie structure and available memory. The StrongARM traverses the trie structure looking at each node's active bit. If the active bit is clear, the router has seen no traffic with the node's SAS since the last cleaning interval, and the node is removed from the trie.

5.3. Synchronization and Architectural Considerations

The StrongARM and the microengine share the same trie data structure, and both processing engines sometimes make updates to the structure. These updates must be properly synchronized to ensure that the trie remains consistent.

Since only the StrongARM creates and deletes trie nodes, there is no possibility that simultaneous creations, deletions, or a combination from the two engines will corrupt the data structure. In some cases, the microengine might read an entry that the StrongARM is about to delete. There is no harm done if this happens because the result is the same as if the microengine had just beaten the StrongARM to the deletion. Furthermore, new trie levels are connected from the bottom up to the root and deletions are performed from the top down to eliminate the presence of invalid data in the structure during structural modifications.

One architectual issue of note is that the microengines and StrongARM do not have a synchronized real-time clock. Our timestamp mechanism depends on time synchronicity between the two components. In its current incarnation, this mechanism has the StrongARM increment two 32-bit words located in the fast, shared scratch memory. One word represents the valid timer and is incremented several times per second. The other represents the legacy timer and can be incremented at a much longer interval, such as once per hour. Timestamps in the data entries are compared with one of these two timers in the scratch memory. Timestamp overflow in the 23- and 24-bit storage spaces are handled by a StrongARM component that infrequently traverses the trie and resets expired entries before resetting the associated scratch-resident timer.

6. PERFORMANCE OF THE IXP ISAVE IMPLEMENTATION

Ideally, every packet going through an iSAVE router should be checked for source address validity. The major goal of porting iSAVE to the IXP router was to allow these checks to be done rapidly enough to achieve hardware router speeds while still providing address assurance. This section presents performance data showing that the implementation succeeds in its performance goals.

We measured the bandwidth achieved by the router with and without the iSAVE functionality in place for packets of varying size. We also measured the number of packets dropped in each circumstance. These comparisons showed how much performance was lost by adding the iSAVE functionality to the IXP router already running standard IPv4 forwarding code. We also measured the same quantities for a direct connection without the IXP in the middle. This comparison shows how much of a performance penalty is suffered merely due to having the IXP in the loop.

The values shown for the IXP were taken when two pairs of nodes were sending packet streams to each other through the IXP. The IXP without iSAVE functionality merely did packet assembly and IPv4 forwarding. With iSAVE, the IXP also checked each packet's source address against the appropriate entry in the incoming trie. The trie had been preset with proper interface values for the four addresses, and none of the packets sent had a forged address. The trie entries remained valid for the entire experiment, so no requests for updates had to be sent. This scenario represents a favorable case for iSAVE, but is not too different from many normal operational scenarios.

Using the IXP at all costs some bandwidth. Depending on packet size, the difference is less than 10%, with larger packets incurring smaller penalties. Fewer larger packets fill the channel, and the IXP processing overheads are primarily per packet, so the costs are higher for smaller packets. Even at 512 byte packets, the IXP achieves a throughput of 80 Mbps.

Adding the iSAVE functionality incurs no noticeable costs. For 1512 byte packets, the IXP augmented with iSAVE even achieves slightly higher throughput, though the difference is not nearly high enough to be statistically significant in these measurements. In essence, the costs of using the hardware at all and of performing IPv4 forwarding dwarf the costs of checking iSAVE's trie to determine if a packet should be dropped.

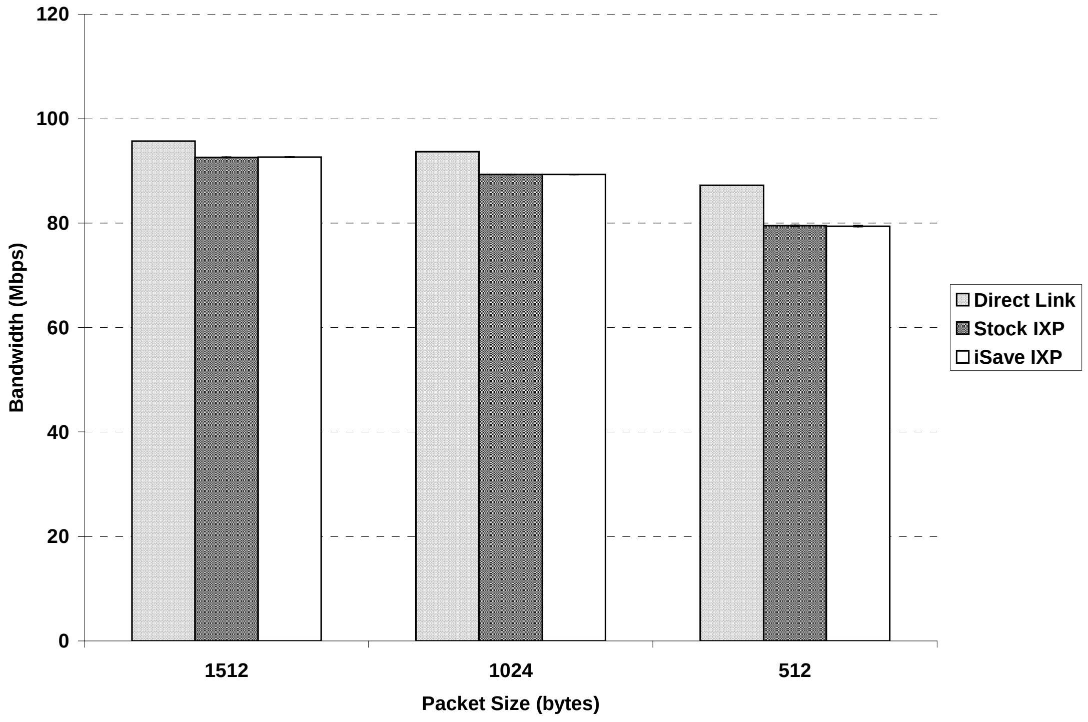


Figure 3. Bandwidth achieved

Part of the reason for reduced throughput when the IXP is involved is that packets are dropped when the router cannot keep up with the data flow. Figure 4 shows the number of packets dropped in the same tests as those described for Figure 3. A direct link suffers almost no drops, except at the smallest packet size tested. The IXP cases suffer more drops, with nearly 10% of all packets dropped for 512-byte packets. The drops suffered almost completely account for the decrease in bandwidth shown in Figure 3. Note, however, that the iSAVE functionality causes no more drops than the standard IXP configuration did. (The apparent difference between the figures for the standard IXP configuration and the iSAVE configuration at a packet size of 512 bytes is statistically insignificant, as is shown by the overlapping 95% confidence interval bars.)

These results do not tell the full story of the performance of the IXP iSAVE implementation, of course. Because the protocol requires control messages to set up its table entries, the performance of the router might be limited by the costs of handling those messages. We measured the StrongARM processor's ability to handle updates and determined that it can handle around 335 iSAVE updates per minute.

Further performance experiments will be necessary to fully characterize the IXP iSAVE implementation's performance in real-world use, but these results suggest that little or no performance is lost on the IXP router by adding the iSAVE functionality. This result suggests, in turn, that other security improvements to routers that have a similar complexity (such as installing filters, putting simple marks in packets, or keeping some basic counts of packets going through the router) will also achieve acceptable performance.

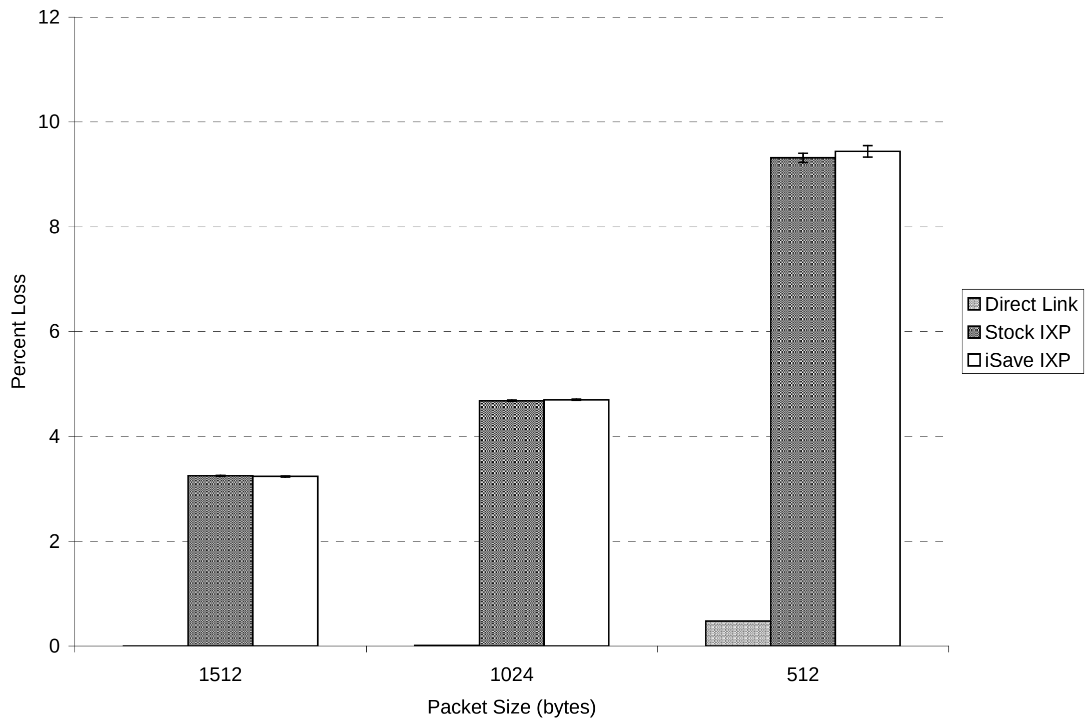


Figure 4. Percent of packets dropped

7. RELATED WORK

Previous approaches to source address validity have been either preventive or reactive. For example, filtering and cryptography-based authentication are preventive approaches. Tracing is primarily reactive.

Much research has focused on applying cryptography in order to guarantee authenticity of packet information, including the source address field (IPsec is one representative at the IP layer [Kent98]). A digital signature on a packet can prove the authenticity of its purported origin. Packet encryption also works, since the recipient will only be able to successfully decrypt a packet if it was encrypted at its true origin. Unfortunately, the high computational overhead of cryptographic operations prevents such approaches from being widely employed on a per-packet basis. This also usually involves complex key exchanges at the Internet scale.

Also widely studied is packet tracing [Bellovin01]. While tracing is a very useful capability, a disadvantage is that it is typically performed after an attack is detected, frequently too late to avoid damage. Moreover it requires fairly complex operations. These difficulties are compounded when the volume of attack traffic is small or the attack is distributed [Park01].

In contrast, filtering is a preventive approach and does not necessarily need cryptographic operations. Ideally, a router should be able to determine the validity of the source address of any packet, as long as that packet is from a protected domain. Filtering based on routing tables is one possible approach; however, it assumes symmetric routing, i.e., the outgoing interface that a router uses to reach a given address, as specified by its routing table, is also the valid incoming interface for packets originating from that address. However, since routing asymmetry on the Internet is common, this approach is unrealistic.

Source-address-based filtering has been supported by router vendors (such as Cisco and 3COM) using static, manual configuration (such as access control lists for Cisco routers) [CERT00]. Martian addresses (loopback address, broadcast address, non-unicast addresses, etc.) can be easily filtered [Baker95]. Particular rules can also be applied. For instance, ingress filtering [Ferguson00] can be enforced at a periphery router by ensuring that packets leaving the domain of a periphery router have a source address from inside the domain, and egress filtering can ensure that packets entering have an outside source address. This kind of filtering, however, does not offer sufficient protection. For instance, research has shown that unless ingress filtering is deployed almost everywhere, nearly arbitrary forgery is still possible [Park01].

Many companies have network processors out in the market. Agere, AMCC, IBM, Motorola, and Vitesse are some of the more well-known names [Glaskowsky01]. These processors can handle the very fast data speeds associated with Internet traffic. We have not studies these processors fully enough to discuss how the iSAVE protocol would be best implemented on them.

Jon Turner's active networks group at Washington University has developed an active networks node [Descaper99]. A network processor attached to the backplane of an ATM router allows packets to be switched without help from the core processor. The active network applications can also be spread over the network processors for load-balancing reasons. We have found no current research into putting source address validity checking into a network processor-based system.

8. CONCLUSIONS

Because the Internet was not designed with security in mind, it is highly vulnerable to any number of attacks, and new attacks are being discovered regularly. Many of the defenses necessary to counter such attacks require some degree of assistance from routers, either at the edges of the network or in the core. Much security research is going forward on the assumption that such assistance will be available in the future.

The nature of security systems is that they are adversarial. Attackers find ways to disable or avoid protections put in place, and thus those protections must often be updated and improved. Using hardware implementations of new security mechanisms is risky, since if an attacker can find a way around them, the feature becomes worthless and the supposedly secure router becomes vulnerable again.

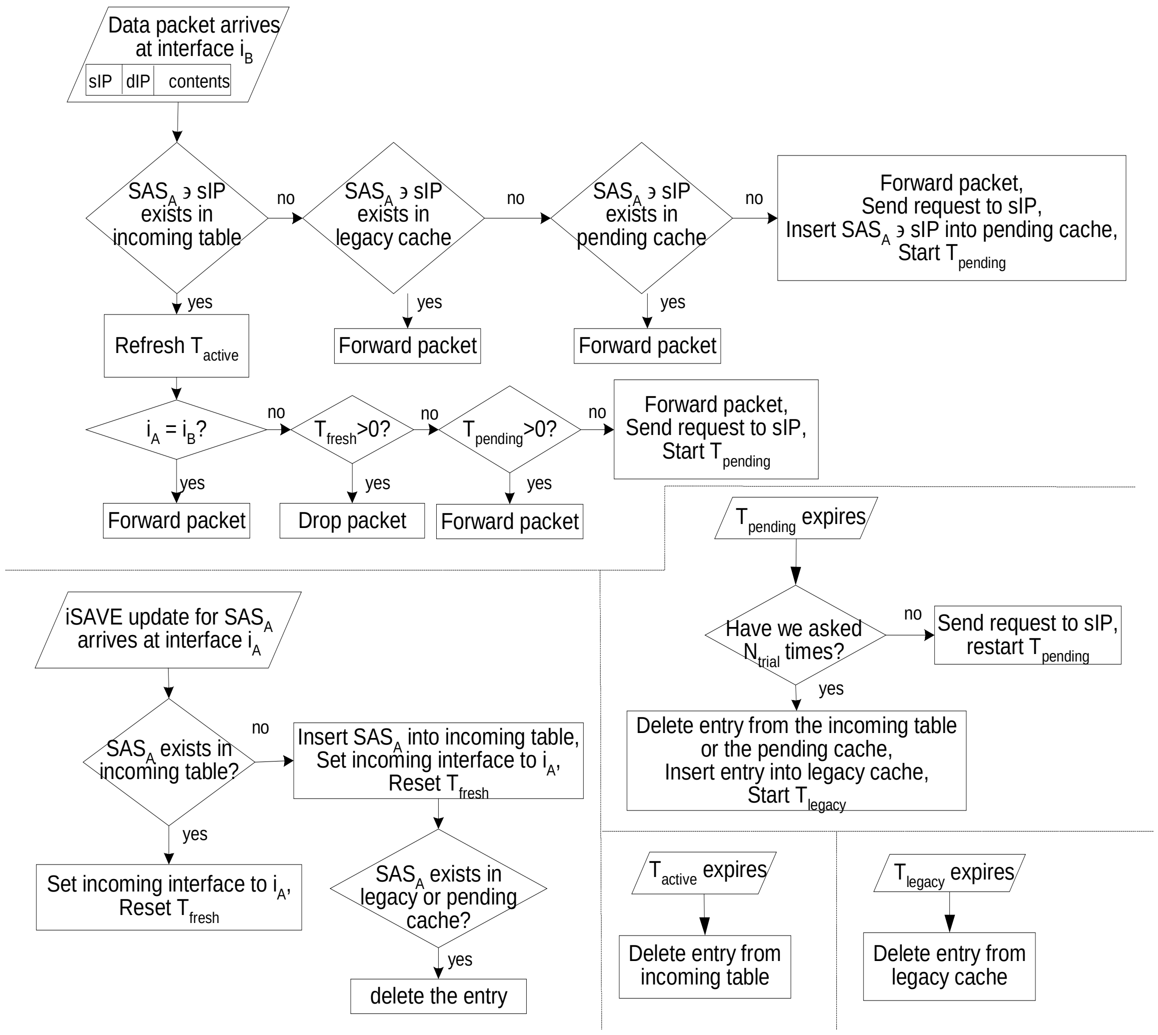
Programmable routers offer a solution to this problem. Since they are far easier to upgrade, new security features can be added without making expensive hardware modifications or replacing the entire unit. But this solution is practical only if a programmable router can perform the necessary security functions at high enough speed to keep up with the pace of This paper shows that one sample programmable router, the Intel IXP, is able to perform one sample network security function in a router at high speeds. The router suffers essentially no performance degradation due to the addition of the iSAVE filtering capabilities. Beyond showing that this particular protocol is practical on this particular router, however, the research suggests that many proposed security features could be practical on many types of programmable routers. The kinds of operations that iSAVE performs are similar to those required by many other router security systems. While the implementation is specific to the IXP, it uses router features that are likely to be present on a wide range of programmable routers.

Thus, this research suggests that those investigating Internet security features that require deployment in Internet routers should, when possible, perform much of their research on a programmable router, rather than relying purely on simulation or software router implementations. This choice will lead to more realistic research and a faster path to deployment for successful Internet security systems.

REFERENCES

[Baker 95] F. Baker, "Requirements for IP Version 4 Routers," RFC 1812, June 1995.
[Bellovin 01] S. Bellovin, M. Leech, and T. Taylor, "ICMP Traceback Messages," Internet Draft, Work in Progress, http://search.ietf.org/internet-drafts/draft-ietf-itrace-01.txt, October 2001.
[Bremford] M. Bremford "Ping O' Death Page, http://www.pp.asu.edu/support/ping-o-death.html.
[Burch 00] H. Burch and W. Cheswick, "Tracing Anonymous Packets to Their Approximate Source," Proceedings of 2000 System Administrators Conference, December 2000.
[CERT 00] Computer Emergency Response Team, "CERT Advisory CA-2000-01 Denial-of-Service Developments," http://www.cert.org/advisories, 2000.
[Descasper 99] D. Descapser, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner, "A Scaleable, High Performance Active Network Node," in IEEE Network, January/February 1999.
[Ferguson 00] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," RFC 2827, May 2000.
[Glaskowsky 01] P. Glaskowsky, "Network Processors Mature in 2001," www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/ , 2001.
[Intel] "Intel IXA Software Developers Kit 2.01 for IXP1200", http://developer.intel.com/design/network/products/npfamily/sdk2.htm
[Johnson 02] E. Johnson and A. Kunze, *IXP 1200 Programming*, Intel Press 2002.
[Kent 98] S. Kent and R. Atkinson, "Security Architectures for the Internet Protocols," RFC 2401, November 1998.
[Li 02] J. Li, J. Mirkovic, M. Wang, P. Reiher and L. Zhang, "SAVE: Source Address Validity Enforcement Protocol", Proceedings of INFOCOM 2002, June 2002.
[Mirkovic 02] J. Mirkovic, Z. Xu, M. Schnaider, J. Li, P. Reiher, and L. Zhang, "iSAVE: Incrementablly Deployable Source Address Validation," UCLA Technical Report no. 020030, 2002.
[Oregon 02] http://antc.uoregon.edu/route-views, 2002.
[Park 01] K. Park and H. Lee, "On the Effectiveness of Probabilistic Packet Marking for IP Traceback Under Denial of Service Attack," Proceedings of INFOCOM 2001, April 2001.
[Savage 00] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback", Proceedings of ACM SIGCOMM 2000, August 2000.
[Stone 00] R. Stone, "CenterTrack: An IP Overlay Network for Tracking DoS Floods", Proceedings of 9th USENIX Security Symposium, August 2000.

APPENDIX A: FLOWCHART OF ISAVE F-ROUTER


Data packet arrives at interface i_B
sIP
dIP
contents
SAS_A э sIP exists in incoming table
no
SAS_A э sIP exists in legacy cache
no
SAS_A э sIP exists in pending cache
no
Forward packet, Send request to sIP, Insert SAS_A э sIP into pending cache, Start T_pending
yes
Refresh T_active
yes
Forward packet
yes
Forward packet
i_A = i_B?
no
T_fresh>0?
no
T_pending>0?
no
Forward packet, Send request to sIP, Start T_pending
yes
Forward packet
yes
Drop packet
yes
Forward packet
T_pending expires
Have we asked N_trial times?
no
Send request to sIP, restart T_pending
yes
Delete entry from the incoming table or the pending cache, Insert entry into legacy cache, Start T_legacy
iSAVE update for SAS_A arrives at interface i_A
SAS_A exists in incoming table?
no
Insert SAS_A into incoming table, Set incoming interface to i_A, Reset T_fresh
yes
Set incoming interface to i_A, Reset T_fresh
SAS_A exists in legacy or pending cache?
yes
delete the entry
T_active expires
Delete entry from incoming table
T_legacy expires
Delete entry from legacy cache