# Secure Name Service: A Framework for Protecting Critical Internet Resources

Yingfei Dong[1], Changho Choi[2], and Zhi-Li Zhang[2]

[1] Dept. of Electrical Engineering University of Hawaii, Honolulu, HI 96822
yingfei@hawaii.edu
[2] Dept. of Computer Science and Engineering University of Minnesota, Minneapolis, MN 55455
{choi,zhzhang}@cs.umn.edu *

**Abstract.** We propose a novel Secure Name Service (SNS) framework for protecting critical Internet resources from unauthorized accesses, denial of service (DoS) and other attacks. The key idea is to enforce packet-origin authentication through *resource virtualization* and utilize dynamic name binding for protecting servers under attacks and improving service availability. Different from static network-level security schemes such as IPsec and VPN, SNS is able to dynamically bind the names of critical resources at the service level, which allows us to actively protect the service resources through a distributed filtering mechanism built on authenticated packet forwarding paths. Our prototype implementation of authenticated packet forwarding components on Pentium 4 Linux machines demonstrates that regular Linux platforms are sufficient to support SNS authenticated packet forwarding on 100Mbps or 1Gbps LANs.

## 1 Introduction

As we become more and more reliant on the Internet for a variety of networking services, the number of network security attacks with the aim to abuse or disrupt such services has also significantly increased. Furthermore, the sophistication of cyber attacks has also increased. The emergence of massive distributed denial-of-service (DoS) attacks is one such example. Unfortunately, because of the *decentralized* and *open* nature of the Internet, it is nearly impossible to protect the entire Internet from cyber attacks. In addition, the cost of such a solution will be economically prohibitive, due to the sheer size of the Internet. It is therefore important to *selectively* secure and protect Internet services that are *critical*, namely, those services that provide significant values.

In this paper we propose a novel approach – *Secure Name Service (SNS)* – to protect critical Internet services from cyber attacks. The proposed SNS

* This work was supported in part by the National Science Foundation (NSF)under the Grant ITR-0085824. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views the NSF.

mechanism serves as a comprehensive "first-line of defense" against unauthorized accesses, intrusions as well as DoS attacks. SNS is built upon and an extension of the standard domain name service (DNS). The basic ideas behind the SNS approach are as follows: A critical Internet service and its associated resources (e.g., servers, databases, etc.) are placed within a (virtual) *secure zone* in the network domain of the service provider, and correspondingly the names of the service and its resources are placed within a *secure name space*, separate from the standard domain name space.

Unlike DNS, where in response to a query for a host name, the corresponding IP address of the host is returned, SNS only answers queries originated from *trusted* network domains, and returns a so-called *secure handle (SH)* instead of an IP address in response to a query for a secure name. In other words, the IP addresses of protected resources such as servers are always concealed from the requesters (even from a trusted domain), and the protected resources are in essence "virtualized" from both trusted and untrusted users. Consequently, a unauthorized user cannot gain access to a protected resource (say, a server) directly via IP address spoofing. Furthermore, legitimate packets from a trusted domain carry *security authenticators* – generated by the trusted domain based on secure handles – and are *verified* before they can enter the secure zone containing the protected resources.

In this paper we describe the proposed SNS architecture which is comprised of two major mechanisms: i) *secure name service* that consists of secure name servers that virtualize protected resources within secure zones, set up security associations (SAs) between domains, and perform secure name resolutions; and ii) *authenticated packet forwarding* that consists of *security checkpoints (SCs)* and *security gateways (SGs)*, which verify security authenticators, filter out illegitimate packets, and map secure handles to the IP addresses of protected resources. In addition to *proactive protection*, we also explicitly incorporate *active monitoring* and *rapid response* mechanisms into our proposed architecture for further securing critical services.

The remainder of this paper is organized as follows. In Section 2, we compare the proposed SNS framework with the related work. In Section 3 we describe the design of SNS naming scheme. We present the design of authenticated packet forwarding components and our experimental evaluation in Section 4. In Section 5, we devise two fast lookup schemes for secure name translation and evaluate their performance through analysis and simulation. We conclude the paper in Section 6.

## 2 Related Work and Discussion

In the SNS framework, we combine name service and network security into a unified framework. We briefly review the related work in naming security, traffic security, entity authentication, and proactive and reactive defense schemes. For naming security, DNSSEC [1, 2] mostly focuses on protecting the authenticity and integrity of DNS databases and DNS responses. Although DNSSEC is indeed

an effective way to avoid DNS forgery, it does not address the issue of protecting services under attacks.

IPsec [3, 4] supports traffic security at only the network layer with several limitations. First, IPsec is a rather heavy-duty mechanism which poses many preliminary requirements that hinder its deployment. Furthermore, the scalability of IPsec is a potential issue because an IPsec server needs to negotiate and maintain a security association for each client connection. Lastly, IPsec focuses on traffic security at the network layer, and does not address the issue of protection of service and active defense for improving service availability. Regular VPNs also suffer from this problem. Similarly, TLS [5] ensures the security at the transport layer and does not address the defense issue.

Kerberos [6] is designed for entity authentication that allows a client and a server to mutually authenticate each other across an insecure network. After the mutual authentication, they are able to negotiate a shared secret to exchange encrypted messages for privacy and data integrity. Kerberos does not address the issue of active defense for improving service availability. Existing mechanisms to deal with DoS attacks are often classified into proactive and reactive approaches. Proactive approaches eliminate packets with forged source addresses, such as ingress filtering, Secure Overlay Service (SOS) [7], Mayday [8], and VPN Shield [9]. Ingress filtering uses known unambiguous traffic information to filter out invalid packets at an ingress point, such as source addresses or destination addresses. Therefore, it is suggested for stub domains and low-rate ingress links, but not for transit domains and high-rate links. Ingress filtering does not preclude an attacker using a forged source address within a legitimate prefix filter range. SOS requires a wide-area overlay infrastructure with a large number of intermediate nodes to filter out attacking traffic. VPN Shield provides a limited capability of reacting to flooding attacks. However, it is built on the static IPsec and requires bandwidth reservation at the ingress links of secure domains.

Reactive approaches for DoS attacks include firewalls, IP traceback [10], link testing, input debugging [11], controlled flooding [12], logging [11], ICMP traceback [13], packet marking [12, 10], aggregate-based congestion control, and so forth. They all require either the coordination of human administrators of related domains or the modification of intermediate routers. The complexity of the coordination and the slow error-prone human actions hinder the deployment of these approaches. Furthermore, these approaches only work when attacks have caused some damage, and are less useful to stop unknown attacks.

Compared to the related work, the proposed SNS shows several salient advantages. First, the SNS framework provides a comprehensive first-line of defense through resource virtualization and dynamic name binding, which allows us to apply different security policies at multiple levels and components to address different security threats. As a result, it enhances the service availability with low management costs. In particular, SNS distributes the security check load over security gateways (SGs) and security checkpoints (SCs) in authenticated packet forwarding, and therefore significantly reduces the security costs at critical servers. SCs are responsible for filtering out ingress attacking traffic, while

SGs mostly emphasize secure-packet translation. Consequently, critical servers can sustain their service performance under attacks. In contrast, existing approaches such as IPsec or TLS does not address this issue. As a result, a critical server could not sustain its performance under attacks because it has to devote itself to intensive security checking. Furthermore, SNS is incrementally deployable as it does not require to have a broad infrastructure in place, and it does not require to replace application software.

## 3 Secure Name Service (SNS)

The main functionalities of the SNS naming system are 1) to authenticate hosts, security gateways, and checkpoints in a domain, and manage corresponding security keys and IDs in order to ensure intra-domain packet authentication between hosts and security gateways (or between gateways and checkpoints); 2) to build security associations (SAs) between SNS servers. An SA includes the IP addresses of corresponding security gateways and secret keys for generating and verifying packet authenticators between domains; 3) to maintain a secure name database for secure name resolutions; 4) to resolve secure name queries from trusted hosts. To support these features, we design the SNS naming system consisting of SNS servers, SNS-aware DNS servers, SH managers at SGs, and stub resolvers at hosts. We refer readers to [14] for the details of the secure name service mechanism and components.

In the SNS naming framework and forwarding mechanism, we add other three identities combining with an IP address to represent a host at different stages of packet forwarding, i.e., *Secure Handle (SH), Host ID* and *External Identity*. We use a 32-bit secure handle ($SH_X$) in a response as the *SNS identity* to represent a destination host $X$ when a packet is sent from a host to an SG. This SNS identity is viewed as a *virtual* IP address by applications, and it is used a forwarding label in the authenticated packet forwarding in a secure zone. When a packet is forwarded from an SG to a host, we use the host IP address to represent the host. Because we hide each host behind an SG, to distinguish each host, we assign a host identifier $H\_ID_X$ to a host $X$. In addition, we define ($SG\_IP_X$, $H\_ID_X$) as its *external identity* to represent $X$ outside its home zone, where $SG\_IP_X$ is the IP address of the SG for this host $X$.

A secure name resolution maps a secure name into an SNS identity (an SH). The basic process of resolving a secure name query is shown in Fig.1. An SNS stub resolver $S_1$ at a host recognizes an SNS query $Q$ for the identity of a secure name $X$, and then forwards this query to its SNS. When this query arrives at $SG_1$, $SG_1$ authenticates this message and then forwards it to $SNS_1$. $SNS_1$ looks up its secure name database and finds the external identity of $X$, i.e., $(SG\_IP_X, H\_ID_X)$. (If $X$ is not in the database, $SNS_1$ will obtain the external identity of $X$ by issuing a secure name query to SNS server $SNS_2$ that manages secure name $X$.) Then $SNS_1$ passes the external identity of $X$ to SH manager $M_1$ at $SG_1$ in a response $R'$. Upon receiving $R'$, $M1$ first checks if the external identity of $X$ is in its SH database. If it is, $M1$ finds $SH_X$ from the database;

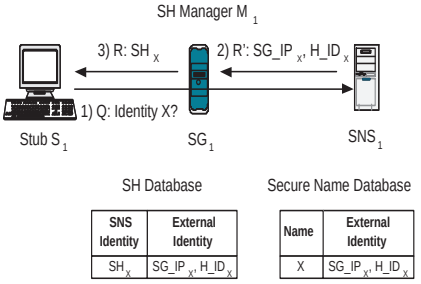



**Fig. 1.** Resolving a query by a local SNS.

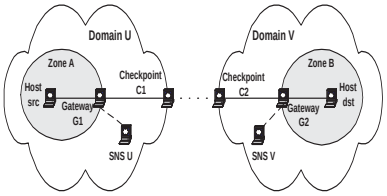



**Fig. 2.** Two SNS-enabled Domains.

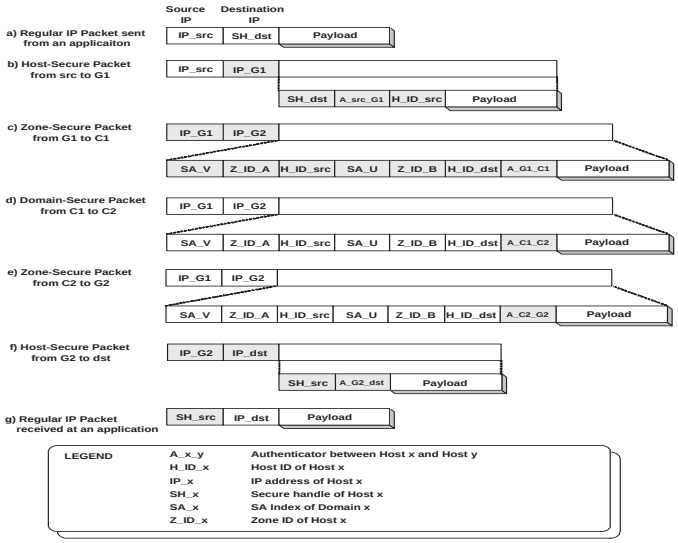



**Fig. 3.** Packets from Host *src* to Host *dst*.

otherwise, $M_1$ inserts an entry into the SH database for this external identity and obtains $SH_X$. Then, $M_1$ sends a response $R$ to $S_1$ with the $SH_X$ as the response to query $Q$.

## 4 Authenticated Packet Forwarding

The secure packet forwarding mechanism consists of secure IP layers at end hosts, security gateways (SGs) of secure zones, and security checkpoints (SCs) of secure domains. We use an example as shown in Fig.2 to explain how the SNS framework achieves the secure communication between Host *src* in Zone $A$ of Domain $U$ and Host *dst* in Zone $B$ of Domain $V$, without revealing their IP addresses. Assume an application on host *src* first obtains a secure handle SH_dst of host *dst*, and it then constructs a regular IP packet using SH_dst as

the destination address, as shown in Fig.3.a. Before this packet is passed the link layer at *src*, it is intercepted by the sIP layer at *src*. The sIP layer recognizes this packet by its secure handle, and then translates it into a host-secure packet, as shown in Fig.3.b. The packet is then forwarded as a regular IP packet. When the packet reaches gateway $G1$ of Zone $A$, $G1$ translates the IP packet into a zone-secure packet, and forwards it to checkpoint $C1$, as shown in Fig.3.c. Based on security parameters between $G1$ and $C1$, $G1$ generates and inserts a *zone authenticator* (*A_G1_C1*) into the packet. As shown in Fig.3.c, the destination host ID *H_ID_dst* and the remote zone ID *Z_ID_B* are also inserted into the packet to ensure this packet is correctly routed to the host *dst*. Moreover, the source host ID *H_ID_src* and the source Zone ID *Z_ID_A* are also inserted into the packet in order to provide sufficient routing information for return packets to be routed back to host *src* when they return to $G1$.

At $C1$, we first check the zone authenticator *A_G1_C1*. If invalid, the packet is dropped. Otherwise, we compute a *domain authenticator A_C1_C2* to replace *A_G1_C1*, as shown in Fig.3.d. We use BGP announcements to direct packet routing between domain $U$ and $V$ such that the above domain-secure packet is forwarded from Checkpoint $C1$ to Checkpoint $C2$ across regular IP networks in between. At $C2$, we first check the domain authenticator of a packet using its remote SA Index *SA_U*. If invalid, the packet is dropped. Otherwise, we then generate a zone authenticator *A_C2_G2*. As shown in Fig.3.e, we replace *A_C1_C2* with *A_C2_G2* in the packet and forward it to $G2$. Upon receiving the zone-secure packet, $G2$ first checks if its zone authenticator is valid. If valid, $G2$ translates the packet into a host-secure packet as shown in Fig.3.f; otherwise, $G2$ drops the packet. Furthermore, $G2$ looks up its remote IP address database to check if it needs to insert a new entry in the database because it needs to remember how to route a return packet from Host *dst* to Host *src*.

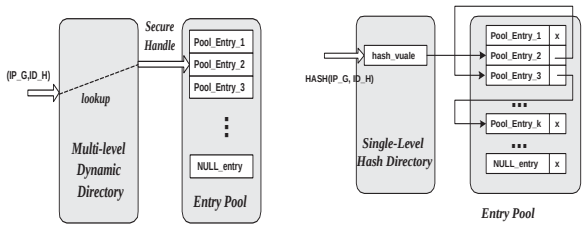
When the host-secure packet arrives at host *dst*, the secure IP layer recognizes it as a secure packet based on the protocol field in its IP header. It first translates the host-secure packet into a regular IP packet, and then puts this new packet into the IP input queue. Consequently, an application at Host *dst* receives a regular IP packet as shown in Fig.3.g.

We have implemented the prototypes of sIP layer, SG and SC on Linux kernel 2.4.20 using Linux Netfilter for evaluating authenticated packet forwarding of SNS. We refer readers to [15] for the details of the implementation and introduce the performance results in the following. Utilizing the time stamp counter (TSC) of Pentium CPUs to directly read CPU clock cycles, we can measure the delay at each step of our implementation in clock cycles. We use three Linux machines such as H1, H2, and H3. H1 and H2 houses a 2GHz Pentium 4 processor,512 MB memory, 8KB L1 cache, and 512KB L2 cache. H3 is a 2.8GHz Pentium 4 processor machine with 1GB memory, 8 KB L1 cache, and 512KB L2 cache.

We summarize the delays at the components of authenticated packet forwarding in Table 1. For the testing of sIP layer, we send 10,000 UDP packets of 1024 bytes over a direct link between H1 and H2. We also use HMAC-MD5 for MAC generations. The overall delay of the sIP layer is 6879 cycles (3.44$\mu$s). We

**Table 1.** Delays of Forwarding Components(in clock cycles)

| | Authenticator Initialization | MAC Check | Secure Packet Translation | MAC Generation | Total | Effective Bandwidth |
|---|---|---|---|---|---|---|
| sIP | 3067 | - | - | 3812 | 6879 | 291 MB |
| SG | - | 4463 | 450 | 3587 | 8500 | 329 MB |
| SC | - | 4455 | - | 3869 | 8324 | 337 MB |




(a) Multi-Level. (b) Single-Level.

**Fig. 4.** Two Types of Address Translation Table.

also measure the effect of sIP on end-to-end bandwidth using *Iperf* from NLANR (www.nlanr.net). On a 100Mbps link, we can achieve a transmission rate of 93.9 Mbps over regular IP and a transmission rate of 91.9Mbps over sIP, which is 98% of the rate using IP.

To evaluate the performance of an SG, we measure the delays of packet authentication (MAC check), secure packet translation, and MAC generation, as shown in the second row of Table 1. We connect host H1 to H2 through H3, which acts as an SG. Again, we send 10,000 UDP packets of 1024 bytes from H1 to H2. We use the similar setting of SG to test H3 as an SC. The results are also shown in the third row of Table 1. The last column in Table 1 shows that our prototype can support a transmission rate around 300 MBps, which is sufficient for a LAN environment with a 100Mbps or 1Gbps link. The experimental measurements on the prototype of sIP layer, SG and SC on regular Linux machines have shown the feasibility of the SNS authenticated packet forwarding schemes.

## 5 Dynamic Table Management at an SG

In the process of secure address translation at an SG, we need to authenticate and translate an incoming secure packet based on its address pair (IP_G, ID_H) or an outgoing packet based on its SH, where IP_G is the 32-bit IP address of a remote security gateway and ID_H is a 16-bit remote host ID. To ensure the correct mapping in both incoming and outgoing directions, we need both an SH and a (IP_G, ID_H) pair of the same flow to point to the same entry in the address table. Different from traditional dynamic table mechanisms, which only access tables through a primary key, we need to use both a (IP_G, ID_H)

pair and an SH to access an address entry. Therefore, we design a two-layer structure to address this issue. At the lower layer, we use an *Address Entry Pool* consisting of address entries, which allows us directly to access address entries using its indexes as SH's. At the upper layer, we build a *dynamic directory* for fast lookups based on a primary key, i.e., (IP_G, ID_H) pair. For fast lookups based on (IP_G, ID_H) pairs, we design a multi-level directory scheme and a single-level directory scheme described in the following. The corresponding structure of entry pools is shown in Figure 4.

We first propose a *Multi-Level Directory Scheme*. Let us denote a 48-bit primary key, a (IP_G, ID_H) pair, as $k_{47}k_{46}\cdots k_0$. At the first level, we use the first 16 bits, $k_{47}k_{46}\cdots k_{32}$, as the index. We use the next 8-bit $k_{31}k_{30}\cdots k_{24}$ as the index of the second-level directory. Similarly, at level three, four and five, we use corresponding 8 bits as the index of subdirectories.

We also design a *Single-Level Hashing Scheme* to reduce potential delays and memory cost in the above scheme, because the total number of hosts is assumed to be smaller than $2^{32}$ and using 48 bits as a primary key may result in an uneven directory tree, which causes unnecessary delays in operations. In this scheme, we need to search through a list by comparing the primary keys of a list to find an SH, because we allow collisions on a table entry. We use hash value $v$ to find the header of a list, where $v = H_1(IP\_G, ID\_H)$, and hash function $H_1$ is implemented using Knuth's multiplication method [16], which can be computed in less than 100 clock cycles on Pentium-4.

We analyze the performance of the above directory schemes in the following. Let us first define the traffic model used in evaluation. Assume we have $N$ clients, each has an on-period $T_i^{on}$ seconds with a rate of $r_i$ packets/sec, and an off-period $T_i^{off}$ seconds, where $1 \leq i \leq N$. Then the average number of active flows generated by clients will be $N_{active} = \sum_{i=1}^{N} \frac{T_i^{on}}{(T_i^{on}+T_i^{off})} \cdot N$.

For a packet $j$, the probability that it belongs to an existing flow $i$ is $P[j \in flow\ i] = \frac{r_i}{\sum_{k=1}^{N_{active}} r_k}$. We assume that an address entry is expired after each on-period. Then we need to insert an address entry for a flow in each on-off cycle. The probability that packet $j$ causes a table insertion for flow $i$ is $P[j\ causes\ an\ insertion] = \frac{1}{T_i^{on} \cdot r_i}$. Therefore, for packet $j$, the probability that it causes an insertion for flow $i$ is $P_{insert}^{(i)} = P[j \in flow\ i] \cdot P[j\ causes\ an\ insertion]$.

We first analyze the performance of the multi-level directory scheme under the above traffic model. Figure 5 shows the lookup algorithm that decides the action for a packet of flow $i$, whose address is fallen into directory entry $e$. Consider level $l$ directory with $2^k$ entries, where $k = 16$ when $l = 1$, and $k = 8$, when $2 \leq l \leq 5$. Let $N_l$ be the current flow population in level $l$ and its sub-directories. We know $N_1 = N_{active}$. Assume client addresses are uniformly distributed across the whole directory, the expected population in the level $l$ is $N_l = \frac{N_1}{2^{16+8\cdot(l-2)}}$, $2 \leq l \leq 5$.

Assume packet $j$ arrived at directory level $l$ is fallen into an entry $e$ with a uniform probability of $\frac{1}{2^k}$. Let $p_0^l = P^l[e = 0]$ be the probability that entry e is not occupied currently (i.e., flag $F = 0$); $p_1^l = P^l[e = 1]$ is the probability that

```
1. if (entry e is empty)
2.   INSERT(i); // insert client i into entry e
3.   return a secure handle;
4. else
5.   if (exact one client is in entry e)
6.     if (i is the same as the client in entry e)
7.       return a secure handle;
8.     else // collision
9.       EXPAND(); // expand a next-level directory
10.      INSERT(i); INSERT(i'); // insert both into the next level
11.      return a secure handle;
12.  else // at least two clients are in entry e
13.    step down into the next level directory.
```

**Fig. 5.** Lookup of Multi-Level Directory

```
1. if H_i(key) ≥ p
2.   index = H_i(key);
3. else
4.   index = H_{i+1} (key)
5. access the entry at the index;
6. search through a overflow list if necessary;
```

**Fig. 6.** Lookup in Linear Hashing.

entry $e$ is currently occupied by a single flow (i.e., flag $F = 1$), and $p_2^l = P^l[e = 2]$ is the probability that entry $e$ is currently occupied by more than one flow (i.e., flag $F = 2$), and thus it is expanded into the next level $l + 1$ (for $l < 5$). Then we have $p_0^l = (1 - \frac{1}{2^k})^{N_l}$, $p_1^l = (1 - \frac{1}{2^k})^{N_l - 1} \cdot \frac{1}{2^k}$, and $p_2^l = 1 - p_0^l - p_1^l$. Because of no collisions in the fifth level, we have $p_0^5 = 1$, $p_1^5 = 0$, and $p_2^5 = 0$. Therefore, the expected delay of inserting a new entry into a directory at level $l$ and its sub-directories, denoted by $D_{insert}^l$, is given recursively by Equation 1.

$$D_{insert}^l = d_{flag} + p_0^l \cdot d_{insert} + p_1^l[d_{compare} + d_{expand} + E_{insert}^{l+1}(i, i')] + p_2^l[d_{down} + D_{insert}^{l+1}] \ . \quad (1)$$

where $d_{flag}$ is the delay to determine the flag value of a directory entry, $d_{insert}$ is the delay to insert client information into an entry, $d_{compare}$ is the delay to compare the destination of a packet with that of an existing entry, $d_{expand}$ is the delay to expand a sub-directory in the next level, $d_{down}$ is the delay to step down into the next-level sub-directory, and $E_{insert}^{l+1}(i, i')$ is the delay to insert two distinct entries, $i$ and $i'$, into a newly-expanded sub-directory at level $l + 1$, as defined in Equation 2.

$$E_{insert}^l(i, i') = \frac{1}{2^{16+8\cdot(l-1)}} E_{insert}^{l+1}(i, i') + (1 - \frac{1}{2^{16+8\cdot(l-1)}}) \cdot 2 \cdot d_{insert} \ . \quad (2)$$

where $2 \leq l \leq 4$. For $E_{insert}^5(i, i') = 2 \cdot d_{insert}$ because no collision occurs at the fifth level. The expected delay of searching an entry at level $l$ and its sub-directories, denoted by $D_{lookup}^l$, is given recursively by Equation 3.

$$D_{lookup}^l = d_{flag} + p_1^l \cdot d_{compare} + p_2^l[d_{down} + D_{lookup}^{l+1}] \ . \quad (3)$$

In summary, for the packets of flow $i$, the expected delay of an address insertion is $D_{insert}^1$, and the expected delay of an address lookup is $D_{lookup}^1$. Then the expected delay of a directory lookup/insertion is thus:

$$D(i) = P_{insert}^{(i)} \cdot D_{insert}^1 + (1 - P_{insert}^{(i)}) D_{lookup}^1 \ . \quad (4)$$

Now let us analyze the expected memory cost in the multi-level directory scheme. First, we always allocate the top level directory with $2^{16}$ entries. Then, for each collision on an entry, we allocate a sub-directory of $2^8$ entries. For each flow $i$,

it may cause an expansion of a sub-directory at level $l+1$ if it is collided with another address entry at level $l$ (i.e., when flag $F=1$), $1 \leq l \leq 4$. The probability that flow $i$ is collided with another entry at level $l$ is $m(i,l) = (\prod_{k=1}^{l-1} p_2^k) \cdot p_1^l$. Therefore, the potential memory cost due to flow $i$ is $m_i = \sum_{l=1}^{4} m(i,l)$. The potential memory cost of $N_1$ flows is denoted as $M$, where $M = \sum_{i=1}^{N_1} m_i$.

We now analyze the performance of the linear hashing directory scheme. Assume we initialize the directory with $\tilde{N}_0$ entries, say $\tilde{N}_0 = 2^8$. Assume we have a perfect hashing function, then the memory cost of the single-level directory for a population of $N_1$ is denoted as $M_{N_1} = \tilde{N}_0 \cdot 2^k$, where $k = \lfloor log_2 N_1/\tilde{N}_0 \rfloor$, such that $2^{k-1} \cdot \tilde{N}_0 \leq N_1 \leq 2^k \cdot \tilde{N}_0$. We only expand the directory after $2^{k-1} \cdot \tilde{N}_0$ collisions.

For each packet, we need to first search the table to check if it has a corresponding entry there. If not, we then insert an address entry. The probability that the address of the packet is hashed into an empty directory entry is $p_0 = P[X=0] = (1 - \frac{1}{2^k})^{N_1}$, while the probability that its address is hashed into an occupied directory entry is $p_1 = 1 - p_0$. The search procedure of linear hashing is shown in Figure 6.

$$D_{lookup} = d_{hash} + d_p + D_{list} \ . \tag{5}$$

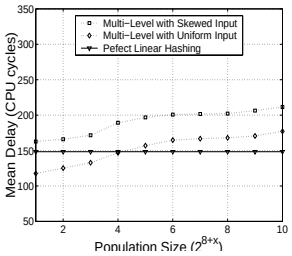
where $d_{hash}$ is the delay of computing the hashing function, $d_p$ is the delay to compare with a splitting pointer $p$, and $D_{list}$ is the expected delay of searching through the overflow list. For a good hashing function, we assume that the average length of the list is less than two. As a result, the upper bound of the delay of searching the list is $D_{list} \leq 1.5 \cdot d_{compare} + 0.5 \cdot d_{next}$, where $d_{compare}$ is the delay to compare the address of the packet with the address in a name entry, and $d_{next}$ is the delay to access the next entry on a list. We then have

$$D_{insert} = p_0 \cdot d_{insert} + p_1 \cdot (d_{hash} + d_p + D_{list} + d_{insert}) \ . \tag{6}$$

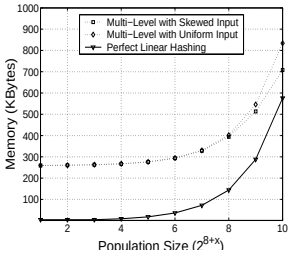
And the expected lookup/insertion delay of packets of flow $i$ is

$$D(i) = P_{insert}^{(i)} \cdot D_{insert} + (1 - P_{insert}^{(i)}) \cdot D_{lookup} \ . \tag{7}$$

We measure the delay of memory read/write and hashing computation in Linux kernel and plug in these parameters into our models. Figure 7 shows the comparison of the multi-level approach with a perfect linear hashing approach. For a uniform distribution of addresses, although the multi-level approach does well for a small population, its delay grows as the population increases. We also test the multi-level approach with a skewed input, in which all address entries are in a single directory entry at the first level and they are uniformly distributed below the first level. In this case, the delay of multi-level approach is increased significantly. While the linear hashing approach keeps a constant delay under the assumption of a perfect hashing function. In addition, the memory cost of the hashing approach is less compared with the multi-level approach, as shown in Figure 7.b.

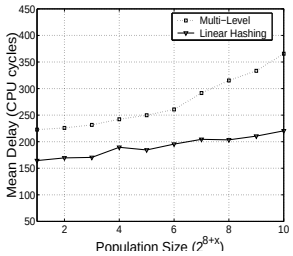



(a) Mean Delay.

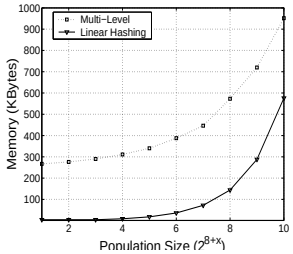



(b) Memory Cost.

**Fig. 7.** Analytical Models.




(a) Mean Delay.




(b) Memory Cost.

**Fig. 8.** Simulations.

We also conduct simulations to evaluate the two schemes. We use a multiplication approach for fast computing hash values, and generate a random set of address lookups. Figure 8.a shows the mean delay of the hashing scheme is significantly better than the multi-level scheme. Figure 8.b shows that the memory cost of the hashing scheme is also better than the multi-level scheme.

## 6 Conclusion and Ongoing Work

We have proposed the SNS framework to protect critical resources from unauthorized accesses and DoS attacks. Through the *resource virtualization* of SNS, we build a distributed filtering scheme to enforce packet-origin authentication. We have described the basic design of the SNS framework, and addressed the performance bottleneck in its authenticated packet forwarding. Based on our prototype on Linux, we have shown the feasibility of implementing SNS on regular Linux machines. We have also designed two fast secure-handle schemes to address the scalability issue in fast address translation.

To fully exploit the advantages of the SNS framework, we face several challenges in the design of the SNS framework, i.e., scalability, reliability, efficiency, and easy deployment. For reliability, we need to protect security gateways from

attacks (such as packet replay and flooding) because these gateways are exposed to attackers. We will address this issue from two perspectives. First, we will evaluate the tradeoffs between computation costs and probabilities that invalid packets penetrate an ingress filtering mechanism using Bloom Filter [17]. Furthermore, we will investigate the effect of reconstructing dynamic packet forwarding paths to defeat attacks. Currently, we are working on these issues and implementing the complete SNS framework for further investigation.

## References


1. R. Arends and et.al, "DNS security introduction and requirements," *Internet Draft, draft-ietf-dnsext-dnssec-intro-03, IETF*, Oct. 2002.
2. G. Ateniese and S. Mangard, "A new approach to DNS security (DNSSEC)," *ACM Conf. on Computer and Communications Security*, 2001.
3. S. Kent and R. Atkinson, "Security architecture for the internet protocol," *RFC2401, Internet Engineering Task Force*, Nov. 1998.
4. D. Harkins and D. Carrel, "The internet key exchange (IKE)," *RFC2409,Internet Engineering Task Force*, Nov. 1998.
5. E. Rescorla T. Dierks, "The TLS protocol," *Internet Draft, draft-ietf-tls-rfc2246-bis-02.txt*, Oct. 2002.
6. B. Neuman and T. Ts'o, "Kerberos: An authentication service for computer network," *IEEE Comminucation Magazine*, Sept 1995.
7. A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services," *In Proc. of ACM SIGCOMM'02*, 2002.
8. David Aderson, "Mayday: Distributed filtering for internet services," *4th Usenix Symposium on Internet Technologies and Systems, Seattle, Washington*, March 2003.
9. R. Ramanujan and et. al., "Organic techniques for protecting virtual private network (vpn) services from access link flooding attacks," *International Conference on Networking'02*, 2002.
10. Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson, "Practical network support for IP traceback," *Proc. of the 2000 ACM SIGCOMM Conference, Stockholm, Sweden*, Aug., 2000.
11. R. Stone, "Centertrack: An IP overlay network for tracking DOS floods," *Proc. of 2000 USENIX Secuirty Symposium*, July, 2000.
12. H. Burch and B. Cheswick, "Tracing anonymous packets to their approximate source," *Unpublished Paper*, Dec. 1999.
13. "IETF ICMP traceback working group," *http://www.ietf.org/html.charters/itrace-charter.html.*
14. Y. Dong, C. Choi, and Z.-L. Zhang, "Design of secure name service," *Techincal Report, CS, UMN*, 2003.
15. C. Choi, Y. Dong, and Z.-L. Zhang, "Implementation of SNS authenticated packet forwarding mechanism," *Techincal Report, CS, UMN*, 2003.
16. Thomas Cormen, Charles Leiserson, and Ronald Rivest, "Introduction to algorithm," *MIT Press, ISBN 0262031418*, 1986.
17. B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM, 13 (7). 422-426.*