A Security Framework for Protecting Traffic between Collaborative Domains

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

[a] *Dept. of Electrical Engineering, Univ. of Hawaii, HI 96822*

[b] *Dept. of Computer Science and Engineering, Univ. of Minnesota, MN 55455*

Abstract

In this paper, we propose a novel Secure Name Service (SNS) framework for enhancing the service availability between collaborative domains (e.g., extranets). The key idea is to enforce packet authentication through *resource virtualization* and utilize *dynamic name binding* to protect servers from unauthorized accesses, denial of service (DOS) and other attacks. Different from traditional static network security schemes such as VPN, the dynamic name binding of SNS allows us to actively protect critical resources through distributed filtering mechanisms built in collaborative domains. In this paper, we present the architecture of the SNS framework, the design of SNS naming scheme, and the design of authenticated packet forwarding. We have implemented the prototype of authenticated packet forwarding mechanism on Linux platforms. Our experimental results demonstrate that regular Linux platforms are sufficient to support the SNS authenticated packet forwarding for 100Mbps and 1Gbps Ethernet LANs. To further improve the performance and scalability, we have also designed and implemented unique two-layer fast name lookup schemes.

Key words: Network Security, Internet Application/Service, Service Availability.

[1] This work was supported in part by the National Science Foundation under the grants ANI-0073819, ITR-0085824, and CAREER Award NCR-9734428. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

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 (DDOS) 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.*

In this paper we propose a novel approach – *Secure Name Service (SNS)* – to protect critical Internet services in collaborative domains (e.g., extranets for business partners) from cyber attacks. As a result, transactions between partners will not be stopped by attacks and servers are not clogged by attacking traffic. The proposed SNS mechanism serves as a comprehensive *first-line of defense* against unauthorized accesses, intrusions as well as DOS attacks. SNS is built upon an extension of the standard domain name service (DNS). The basic ideas behind the SNS approach are as follows: A critical Internet service for trusted business partners 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* collaborative 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, an 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 key mechanisms: i) *secure name service*, supported by secure name servers that virtualize protected resources within secure zones, set up security associations (SAs) between domains, and perform resolutions for secure names; and ii) *authenticated packet forwarding*, supported by *security checkpoints (SCs)* , *security gateways (SGs)*, and *secure IP layer (sIP)*, 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 response mechanisms into the proposed architecture for further ensuring the availability of critical services.

We will first introduce the SNS naming service, supported by *SNS servers, SNS-aware DNS servers, SH managers,* and *SNS stub resolvers*. This mechanism is in charge of establishing secure associations between SNS domains, managing the key distribution within an SNS domain, supporting secure name resolutions, and maintaining the mapping between different identities in authentication.

We then present the design of the SNS authenticated packet forwarding, supported by sIP layers at hosts, SGs of secure zones, and SCs of secure domains. An sIP layer at a host authenticates and translates regular IP packets into SNS packets, and vice versa. An SG authenticates secure packets from hosts, other SGs, or SCs of the same domain and then forwards these packets to corresponding parties based on their security mapping. An SC authenticates packets from SCs of other secure domains or from SGs of its local domain. We have implemented prototypes of these components in Linux Kernel 2.4.20 and evaluated their performance through experiments. The performance and scalability of SGs are the critical issues in the SNS forwarding mechanism because

SGs need to perform a secure name translation for each packet. To address these issues, we further design and implement two fast lookup schemes and evaluate their performance through analysis, simulations and experiments.

The SNS framework exhibits several unique characteristics. Different from traditional *static* network-layer security schemes such as VPN, the SNS framework combines name service and network-layer security into a unified framework to protect critical service through resource virtualization and *dynamic* name binding. Different from traditional authentication schemes such as Kerberos, the SNS framework provides an *active* defense mechanism against various attacks at the network level via packet filtering and adaptive forwarding paths. In particular, the dynamic naming binding of SNS allows us to take advantage of multiple routes in SNS domains to ensure the availability of services even when some security components are clogged by attacking traffic. Different from previous attack prevention schemes such as SOS and Onion Routing that require relative large-scale infrastructures, the SNS framework can be deployed incrementally in a domain-by-domain fashion.

SNS has its limitations. First, it focuses on packet authentication. Since most applications in collaborative environments employ various security techniques (e.g., TLS) to address the issues of data integrity, confidentiality, and non-repudiation, we emphasize packet authentication to mitigate attacks at the network layer. Furthermore, because we have limited resources in trusted SNS domains to deal with flooding attacks on SCs and SGs, to further enhance service availability, we may have to employ resources from a third-party to build a protection hierarchy for restricting attack traffic from untrusted domains to these points. In addition, SNS does not address the issue of entity authentication in a domain. Instead, it uses existing approaches such as Kerberos for this purpose.

The remainder of this paper is organized as follows. In Section 2, we first present the architecture and components of the SNS framework, and then discuss related work. In Section 3 we describe the design of SNS naming scheme.

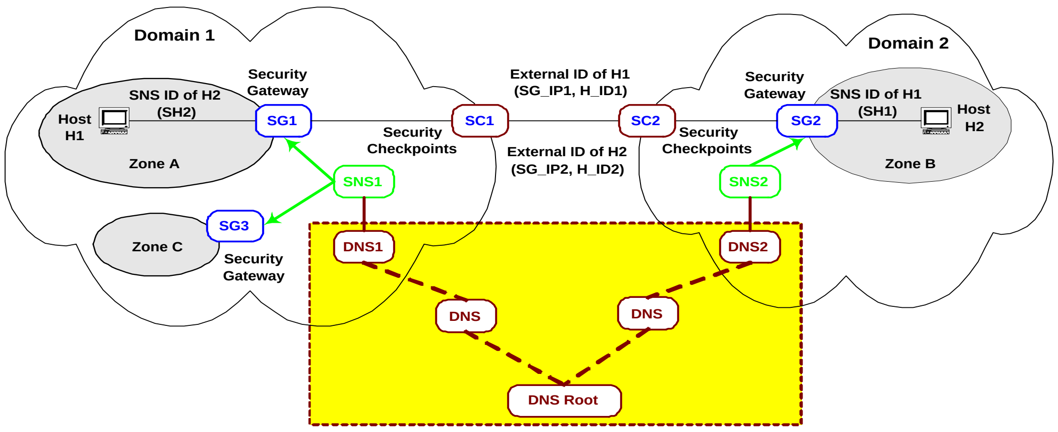


Fig. 1. SNS Framework.

We present the design of authenticated packet forwarding components and the prototype of these 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, simulation and experiments. We conclude the paper in Section 6.

2 SNS Architecture and Components

To protect critical resources from unauthorized accesses and DOS attacks, we need multiple levels of security mechanisms to thwart different security threats, e.g., packet replay, flooding, or IP spoofing. In this paper, we propose the SNS framework as the first-line of defense to filter out invalid packets and actively protect to critical resources. Fig.1 shows the setting of the SNS framework for two collaborative domains. The secure name space consists of secure domains, and each secure domain is comprised of an *SNS server*, several *secure zones* and a number of *security checkpoints (SCs)*. An SNS server manages all secure zones and SCs in a domain. Each secure zone has one or more *security gateway(s) (SGs)* which are responsible for secure packet forwarding for the hosts of the zone.

As shown in Fig.1, we attach an SNS server to a leaf DNS server of the DNS tree, e.g., SNS_1 is attached to DNS_1 and SNS_2 is attached to DNS_2. For secure communication between these two domains, we first build a security association (SA) between them using their SNS servers. Based on this SA, these two domains are able to resolve secure names and authenticate packets from each other by inserting a *packet authenticator* into each packet. For inter-domain packets across insecure networks, we use SCs at the borders of SNS domains to validate them based on their inter-domain authenticators. For packets within an SNS domain, we use SGs to validate them based on their intra-domain authenticators. In addition, each host in a zone authenticates itself to an SNS server before it uses an SG for secure communications.

Through resource virtualization and packet authentication, SNS is capable of protecting critical servers from unauthorized accesses and malicious flooding attacks. The key idea of SNS is to *virtualize the identities of critical resources* in SNS-enabled domains by concealing their IP addresses through secure name service. Different from a regular DNS name resolution that returns a static IP address as the identity of a host, a secure name resolution returns a 32-bit *secure handle (SH)* as the identity of a critical host. This SH is mapped to the real IP address of the host in the SNS framework by SGs, and the IP address is only known to the SNS server and associated SGs. Because this virtualization allows us to decouple the static IP binding, we can not only protect critical hosts from attacks originated from untrusted hosts, but also dynamically adjust the binding to defeat attacks originated from compromised trusted hosts in real-time.

The packet authentication in the SNS forwarding path allows us to build multiple defense mechanisms along the path and apply various security policies at SGs and SCs. When a client at a remote domain exchanges packets with a critical server via a secure handle, packets are authenticated by SGs and SCs on the path between the client and the server. These SGs and SCs filter out invalid packets based on packet authenticators and actively take actions against attacks. In the meantime, they also monitor traffic in order to detect intrusions and brute-force DOS attacks. They can also be used by sophisticated intrusion detection systems to identify and isolate compromised trusted hosts.

In addition to packet authentication, the SNS framework also helps us distribute security check load along packet forwarding paths such that critical services are not clogged due to the considerable security-check load on servers under heavy attacks. Furthermore, the dynamic name binding of SNS allows us to build a distributed filtering mechanism within secure domains. We can choose a different packet forwarding path between two domains when one ingress SC is dragged down by attacking traffic while another SC is normal. Utilizing this dynamic mechanism, we are able to actively adapt to different attack patterns to enhance service availability. We will study the issue of maximizing the service availability, given a set of SCs between two domains. Furthermore, we will also study the issue of building an SNS overlay network such that SNS packets can traverse multiple SNS domains before reaching their destinations, in order to further exploit different paths between SNS domains.

2.1 Related Work and Discussion

In the following, 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. It uses the Public Key Infrastructure (PKI) to generate digital signatures for the authentication of the origin and integrity of DNS queries/responses. Although DNSSEC is indeed an effective way to avoid DNS forgery, it does not address the issue of protecting services under attacks. VPNs based on IPsec [3,4] or L2TP are common approaches used to address the traffic security issue for extranets. TLS [5] ensures the security at the transport layer. Kerberos [6] is designed for entity authentication that allows a client and a server to mutually authenticate each other across an insecure network. However, VPNs, TLS, and Kerberos do not address the issue of service protection and active defense for ensuring 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 (RFC2827), 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.

Reactive approaches for DOS attacks include firewalls, IP traceback [10], link testing, input debugging [11], controlled flooding [12], logging [11], ICMP trace-back [13], packet marking [12,10], aggregate-based congestion control, etc. 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 effectiveness of these approaches. Furthermore, these approaches only work when attacks have caused some damage, and are less useful to stop unknown attacks.

3 Secure Name Service (SNS)

We present the design of secure name service in this section. The main features of the SNS naming system are 1) to build security associations (SAs) between SNS servers, where an SA includes the IP addresses of corresponding security gateways and secret keys for packet authentication between domains; 2) to resolve secure name queries from trusted hosts; 3) to maintain a secure name database for secure name resolutions; 4) to authenticate hosts, security gateways, and checkpoints in a domain, and manage corresponding security keys and identities in order to ensure intra-domain packet authentication between hosts and gateways (or between gateways and checkpoints).

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. Within an SNS domain, we use an SNS server to authenticate all parties in the domain and manage their key exchanges. We also use the SNS server to maintain a secure name database and resolve secure name queries for the domain. We further use SNS-aware DNS servers to help SNS servers to set up SAs between domains in order to perform cross-domain secure name resolutions and packet exchanges. In addition, we use SNS stub resolvers at hosts and SH managers at SGs to recognize, authenticate, and forward secure queries and responses to/from SNS servers. Lastly, we use SNS servers and SH managers to ensure the correct mapping between different identities along packet forwarding paths. In the following, we first introduce key concepts used in SNS and then present the details of these components.

3.1 Secure Name Convention and SNS Identities

In order to facilitate a smooth transition of existing applications from the DNS name space into the SNS name space, we choose the following approaches. First, we let SNS use the same query interface as DNS such that no changes are required for running these applications in the SNS name space. Furthermore, to distinguish secure name queries and DNS name queries at the same interface, we define an *secure naming convention* based on the DNS naming scheme: For a host with a DNS name $x.y.z.w$, we define its SNS domain name as $x\_sec.y.z.w$, by replacing the bottom label x with x_sec. As a result, we can easily migrate a host from the DNS name space into the SNS name space and support applications to access both the secure and the regular name space at the same time in the transition process.

In the SNS naming framework and forwarding mechanism, we define 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*. Because SNS uses the same query interface as DNS, we only have a 32-bit field in a response of a secure name query. Therefore, we use a 32-bit secure handle (SH_X) in a response as the *SNS identity* to represent a destination host X at an SG when a packet is sent from a host to the SG.

This SNS identity is viewed as a virtual IP address by applications, and it is used in the authenticated packet forwarding in a secure zone from a host to an SG. 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 represent each host at the SG, we also assign a host identifier H_ID_X to a host X. Using this host ID, we further define a pair (SG_IP_X, H_ID_X) as the external identity of host X outside its home zone, where SG_IP_X is the IP address of the SG for host X.

3.2 Components of SNS Naming System

SNS server SNS servers are the key components in the SNS naming framework, which perform in the control functionalities of the SNS framework, such as building cross domain SAs, maintaining secure name databases, resolving secure name queries, authenticating all parties in a domain and managing their key exchanges. In the following, we focus on the establishment of SAs and introduce secure name resolution in Section 3.3. We leave the details of intra-domain key management and secure name database in a technical report [14].

In order to establish trust relationship between SNS domains, we assume that an SNS server i obtains a certificate C_i from a *trusted third party (TTP)* through a public key system such as PKI. The C_i includes its name SNS_i and its public key KU_i. Consequently SNS_i is able to use its private key KR_i to sign data exchanged with other SNSs over insecure networks. We denote the *security parameter* of an SNS server i as A_i, where $A_i = \{SG\_IP_i, H\_ID_i, KU_i, Y_i\}$; ($SG\_IP_i$, H_ID_i) is its external identity; KU_i is its public key; and Y_i is its Diffie-Hellman public value; SG_IP_i is the IP address of its SG; H_ID_i is its host ID at its SG. When SNS_i needs to set up an SA with another SNS, it uses KR_i to sign its security parameters $KR_i(A_i)$, and send this signature with its C_i and A_i to another SNS_j. Upon receiving this message, SNS_j verifies the signature using C_i and A_i in the message. After two SNSs validate each other's signature, they compute a shared secret key based on exchanged Diffie-Hellman public values and then use this key for their SA. For ease of discussion, we assume all SNS servers have chosen the same Diffie-Hellman parameter a and q, where q is a large prime and a is a prime root of q. In the following, we introduce a detailed protocol for exchanging security parameters with the help of SNS-aware DNS servers.

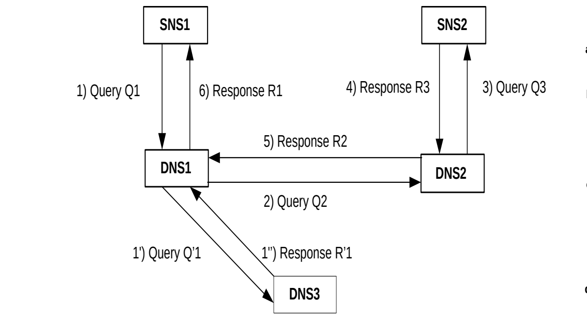
SNS-aware DNS server We extend a leaf DNS server into an SNS-aware DNS server such that the SNS service can utilize the DNS service as a boot-strap point for basic naming service. Utilizing the recursive service at leaf DNS servers, SNS servers are able to exchange security parameters for building inter-domain SAs without revealing their IP addresses. As shown in Fig.1, we attach an SNS server to a leaf DNS server, where DNS_1 is the authoritative DNS name server for domain 1 and SNS_1 is its SNS name server, and DNS_2 is the authoritative DNS name server for domain 2 and SNS_2 is its SNS name server. We make two minor changes on a DNS server. First, we add a new DNS resource record of type RR_{SNS}, in which the RDATA field [RFC1034] is used to pass the security parameter of an SNS server. Second, we add a new type of DNS query and response, named T_{SNS}. Corresponding to this type of query/response, we add a few operations utilizing the recursive service at leaf DNSs, as illustrated in Fig.2. We present the protocol in the following.

(1) When SNS_1 needs to set up an SA with SNS_2, SNS_1 sends a DNS query Q_1 of type T_{SNS} to its DNS server DNS_1. The QNAME of Q_1 includes the name of DNS_2, where DNS_2 is the authoritative DNS server of SNS_2. The additional section of Q_1 includes a resource record of type RR_{SNS}, whose RDATA field holds A_1, the security parameter of SNS_1. The header of Q_1 has the recursive desired (RD) bit set for demanding DNS_1 to perform a recursive service.

(2) During the recursive service for Q_1, DNS_1 first finds the IP address of DNS_2 through standard DNS service, shown as Q'_1 and R'_1 (or multiple iterative queries). Then DNS_1 generates a DNS query Q_2 of type T_{SNS} to DNS_2, in which the QNAME is a NULL string, the recursive desired (RD) bit is set, and A_1 is passed in the additional section.

(3) When DNS_2 recognizes Q_2 of type T_{SNS}, it sends a query Q_3 to SNS_2, passing A_1 in the additional section of message.

(4) From Q_3, SNS_2 receives A_1. Then SNS_2 sends a DNS response R_3 of type T_{SNS} to DNS_2, including its security parameters A_2.



```
1)  Query Q1 : HEADER={RD=1, QDCOUNT=1, ARCOUNT=1},
    QUESTION={QNAME=”DNS2”, QTYPE=”T_SNS”}
    ADDITIONAL={NAME=”SNS1", TYPE=”T_SNS”, RDATA=”SG_IP1,H_ID1,Y1,KR1(SG_IP1, H_ID1, Y1),C1”}
      1')  Query Q’1 : HEADER={QDCOUNT=1}, QUESTION={QNAME=”DNS2”, QTYPE=”NS”}
      1”)  Response R’1 : HEADER={ANCOUNT=1}, ANSWER={NAME=”DNS2”, RDATA=“IP of DNS2”}
2)  Query Q2 : HEADER={RD=1, QDCOUNT=1, ARCOUNT=1},
    QUESTION={QNAME=””, QTYPE=”T_SNS”}
    ADDITIONAL={NAME=”SNS1", TYPE=”T_SNS”, RDATA=”SG_IP1,H_ID1,Y1,KR1(SG_IP1, H_ID1, Y1),C1”}
3)  Query Q3 : HEADER={QDCOUNT=1, ARCOUNT=1},
    QUESTION={QNAME=””, QTYPE=”T_SNS”}
    ADDITIONAL={NAME=”SNS1", TYPE=”T_SNS”, RDATA=”SG_IP1,H_ID1,Y1,KR1(SG_IP1, H_ID1, Y1),C1”}

4)  Response R3 : HEADER={ANCOUNT=1},
    ANSWER={NAME=”SNS2", TYPE=”T_SNS”, RDATA=”SG_IP2, H_ID2, Y2, KR2(SG_IP2, H_ID2, Y2),C2”}
5)  Response R2 : HEADER={RD=1,ANCOUNT=1},
    ANSWER={NAME=”SNS2", TYPE=”T_SNS”, RDATA=”SG_IP2, H_ID2, Y2, KR2(SG_IP2, H_ID2, Y2),C2”}
6)  Response R1 : HEADER={RD=1,ANCOUNT=1},
    ANSWER={NAME=”SNS2", TYPE=”T_SNS”, RDATA=”SG_IP2, H_ID2, Y2, KR2(SG_IP2, H_ID2, Y2),C2”}
```

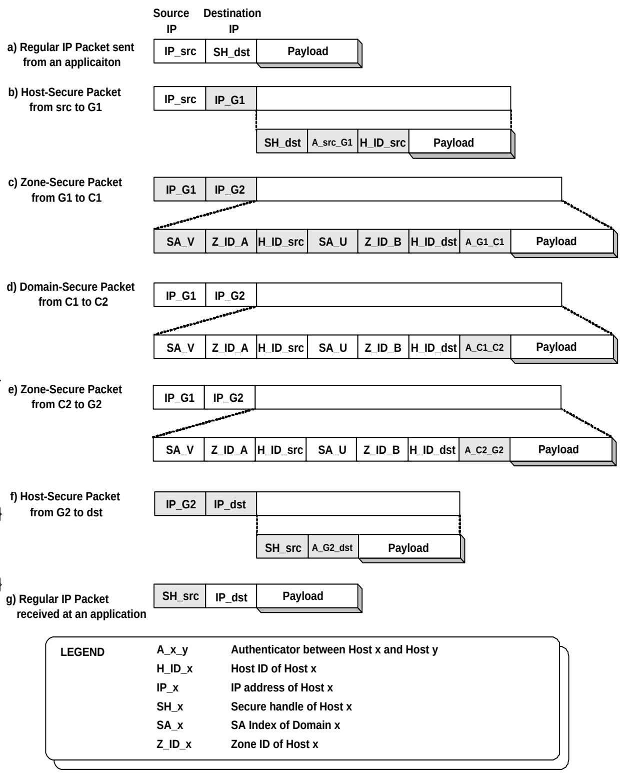

Source IP
Destination IP
a) Regular IP Packet sent from an applicaiton
IP_src
SH_dst
Payload
b) Host-Secure Packet from src to G1
IP_src
IP_G1
SH_dst
A_src_G1
H_ID_src
Payload
c) Zone-Secure Packet from G1 to C1
IP_G1
IP_G2
SA_V
Z_ID_A
H_ID_src
SA_U
Z_ID_B
H_ID_dst
A_G1_C1
Payload
d) Domain-Secure Packet from C1 to C2
IP_G1
IP_G2
SA_V
Z_ID_A
H_ID_src
SA_U
Z_ID_B
H_ID_dst
A_C1_C2
Payload
e) Zone-Secure Packet from C2 to G2
IP_G1
IP_G2
SA_V
Z_ID_A
H_ID_src
SA_U
Z_ID_B
H_ID_dst
A_C2_G2
Payload
f) Host-Secure Packet from G2 to dst
IP_G2
IP_dst
SH_src
A_G2_dst
Payload
g) Regular IP Packet received at an application
SH_src
IP_dst
Payload
LEGEND
A_x_y Authenticator between Host x and Host y
H_ID_x Host ID of Host x
IP_x IP address of Host x
SH_x Secure handle of Host x
SA_x SA Index of Domain x
Z_ID_x Zone ID of Host x

Fig. 2. Process of Exchanging Security Parameters between two SNSs.

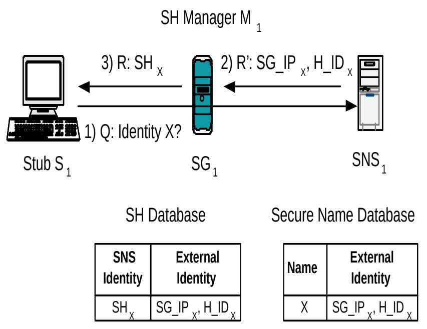
Fig. 3. Packet Formats at each steps from Host *src* to Host *dst*.

(5) When DNS_2 receives R_3, it sends a response R_2 to DNS_1 with A_2.

(6) From R_2, DNS_1 obtains A_2 and passes them in a response R_1 to SNS_1.

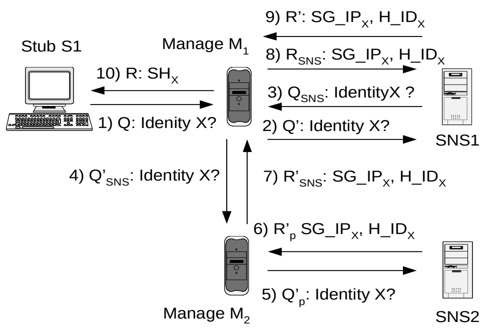
Now SNS_1 and SNS_2 are able to verify each other's security parameters, generate their shared Diffie-Hellman secret keys for their SA, and install their shared secret keys at corresponding SCs. Consequently SNS_1 and SNS_2 are able to exchange their SNS queries and responses through this SA and their external identities.

SNS stub resolver We replace a standard DNS stub resolver with an SNS stub resolver at a client host. This SNS stub resolver has the same interface *gethostbyname()* as a standard DNS stub resolver. While it forwards regular DNS queries to a DNS server as a DNS stub resolver, it is also able to recognize a query for a secure name based on the secure naming convention, and forwards the query to its SNS server for secure name resolution. In other words, this resolver acts as the entrance of the secure name space with no changes



SNS Identity	External Identity
SH_X	SG_IP_X, H_ID_X

Name	External Identity
X	SG_IP_X, H_ID_X



9) R': SG_IP_X, H_ID_X
Stub S1
Manage M_1
8) R_{SNS}: SG_IP_X, H_ID_X
10) R: SH_X
3) Q_{SNS}: IdentityX ?
1) Q: Idenity X?
2) Q': Identity X?
SNS1
4) Q'_{SNS}: Identity X?
7) R'_{SNS}: SG_IP_X, H_ID_X
6) R'_p SG_IP_X, H_ID_X
5) Q'_p: Identity X?
Manage M_2
SNS2

Fig. 4. Resolving a query via a local SNS. Fig. 5. Resolving a query across domains.

required in the current DNS and applications. A stub resolver obtains an SH of its SNS from a Secure IP layer[2] at the host, and forwards secure name queries to UDP/TCP port 53 of its SNS. When a response of a query arrives a stub resolver, it passes the SH in the response as an IP address back to an application.

SH Manager An SH manager maintains an SH database for all secure names at an SG such that the authenticated forwarding mechanism at the SG is able to map SHs to their external identities or IP addresses of local hosts in secure name translations. We use a cache-only SH database at an SG, which is first initialized by an SNS server with local secure names and then populated by cached remote names. Since this database is searched for each secure packet translation at an SG, we must ensure fast lookups in this database. We develop fast lookup mechanisms in Section 5.

3.3 SNS name resolution

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.4. 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,

[2] A secure IP layer is a component of the SNS authenticated forwarding mechanism introduced in Section 4, which is configured with the SH of an SNS.

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, as depicted in Fig.5. We refer readers to [14] for the details of this process.) 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; 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. We are currently implementing this SNS naming framework. We have implemented the authenticated packet forwarding mechanism as introduced in the next section.

4 Authenticated Packet Forwarding

4.1 Design of Authenticated Packet Forwarding

The secure packet forwarding mechanism consists of secure IP (sIP) layers at end hosts, security gateways (SGs) of secure zones, and security checkpoints (SCs) of secure domains. As shown in Fig.6, domain U has an SC $C1$ and an SG $G1$ that is responsible for secure name translations of secure zone A. When a client needs to access a remote secure host *dst*, it first obtains the secure handle SH_{dst} through a secure name resolution. It then uses this SH as the destination IP address for the packet sent to $G1$. $G1$ forwards a packet to the secure destination based on the SH.

An sIP Layer at a host is a small patch to the regular IP layer. When initialized, this layer authenticates itself to an SNS server using a Kerberos-like mechanism based on a pre-configured secret between the host and the SNS. As a result, it obtains a host ID, a host key, the IP address of its SG and SH_{SNS} from the SNS. It uses these parameters for secure communications and secure name queries. For outgoing traffic, the sIP layer intercepts an out-bound regular IP packet destinated to a secure handle, translates it into a *host-secure IP packet*, and then forwards this packet to an SG. For incoming traffic, the sIP layer captures an in-bound host-secure IP packet from an SG and checks

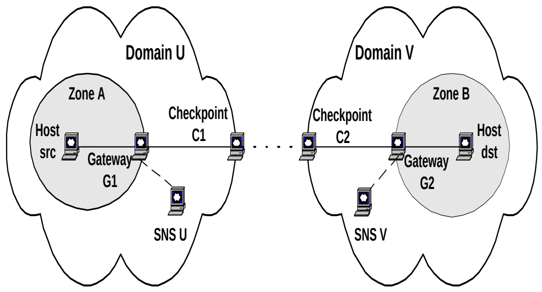


Fig. 6. Two SNS-enabled Domains.

its authenticator. If it is invalid, the packet is dropped. Otherwise, the packet is translated back to a regular IP packet and passed to the transport layer at the host.

An SG of a secure zone forwards host-secure IP packets to their destination gateways based on their secure handles, or vice versa. A host-secure packet from a host to a gateway has a host-gateway authenticator based on their shared keys. For outgoing traffic, when a host-secure packet arrives at an SG, the SG first validates its origin via the host-gateway authenticator. If invalid, the packet is dropped. Otherwise, the gateway performs a *Secure Packet Translation (SPT)*, in which the SG first uses the secure handle to look up the destination address and keys, and then translates the packet into a *zone-secure packet* and forwards it to the destination gateway. Notice that the destination of a zone-secure packet is a remote SG. When a destination gateway receives a zone-secure packet, it first checks its authenticator. If invalid, the packet is dropped. Otherwise, the packet is translated into a host-secure packet, and then forwarded to the destination host. An additional feature of an SG is to monitor and report suspicious host activities. As a result, we can detect and isolate compromised hosts at SGs.

An SC authenticates ingress or egress secure packets. Using Border Gateway Protocol (BGP) announcements, we can control the routes of egress/ingress packets to be routed to chosen checkpoints. An egress zone-secure packet is forwarded from an SG G to an SC C, and it has a zone authenticator generated using the shared keys between G and C. If C finds that the authenticator of a packet is invalid, the packet is dropped. Otherwise, it translates the packet into a *domain-secure packet* by replacing its zone authenticator with a domain authenticator, and forwards this packet to the ingress checkpoint C' of the destination domain. If C' finds that the domain authenticator of a packet is invalid, the packet is dropped. Otherwise, C' translates the packet into a zone-secure packet by replacing its domain authenticator with a zone authenticator. It then forwards the packet to the destination gateway. In the SNS framework, we use border routers to route packets to SGs through SCs such that we can use SCs to filter out invalid packets to SGs and distribute the security check load along the forwarding path.

Illustration of Authenticated Packet Forwarding We use an example as shown in Fig.6 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. The packet has a source IP address of *IP_G1* and a destination IP address of *IP_G2*. 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 $G1$.

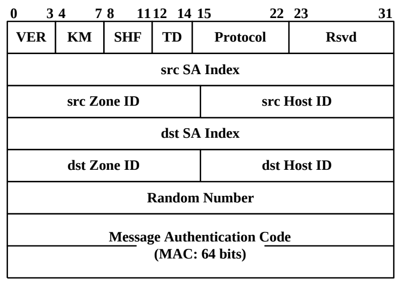
We use BGP announcements to influence the routing tables in domain U such that the above zone-secure packet from $G1$ to $G2$ is forwarded to Checkpoint

$C1$. 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. Similarly, 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, $G2$ does a reverse SPT by translating the packet into a host-secure packet as shown in Fig.3.f; otherwise, $G2$ drops the packet. Furthermore, $G2$ looks up its SH 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.

To be practical, we must address the issue of fast packet authentication, translation and forwarding as well as the scalability of SGs in supporting a large number of hosts. In Section 4.2, we evaluate the performance through our prototype on Linux and present the detailed costs of these components. In addition, we present fast SH lookup mechanisms in Section 5.

4.2 Prototype and Experimental Evaluation

We have implemented the prototypes of sIP layer, SG and SC in Linux kernel 2.4.20 using Linux Netfilter for evaluating SNS authenticated packet forwarding. We refer readers to [15] for the details of the implementation and present the performance results in the following.



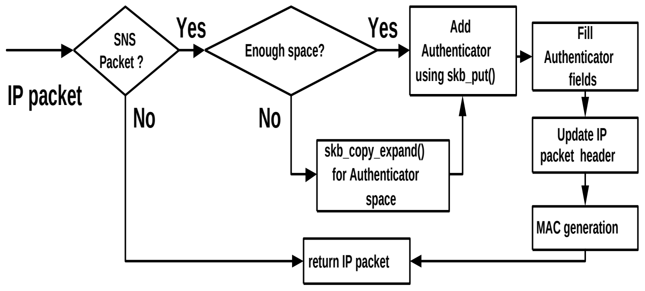

IP packet
SNS Packet ?
Yes
No
Enough space?
Yes
No
Add Authenticator using skb_put()
Fill Authenticator fields
Update IP packet header
MAC generation
skb_copy_expand() for Authenticator space
return IP packet

Fig. 7. Format of Authenticator

Fig. 8. Flow Chart of sIP Procedure

Table 1
Delays of MAC Generation (in clock cycles)

	With precomputed key schedules	Without precomputed key schedules
Blowfish-cbc-mac	2328	129342
Aes-cbc-mac	2738	12708
Hmac-md5	-	3812
Hmac-sha1	-	8368

We insert a 32-byte packet authenticator (as shown in Fig.7) at the end of each SNS packet. The first four bytes are used for control bits, in which bit 0 to 3 indicates the version of SNS protocol, bit 4 to 7 represents the key management protocol, bit 8 to 11 is used for choosing different Message Authentication Code (MAC) generation functions, bit 12 to 14 indicates the direction of the packet (from a host to an SG, from an SG to an SC, from an SC to an SG, or from an SG to a host), bit 15 to 22 is a copy of the protocol field of an original IP packet, and the rest of bits are reserved for future use. The next eight bytes are related to the source host, including a four-byte source SA index, a two-byte source zone identifier, and a two-byte source host identifier. Similarly, the following eight bytes are related to the destination host, including a destination SA index, a destination zone identifier, and a destination host identifier. The next four-byte is a random number for preventing packet replay attacks. The last eight-byte is the MAC value of this packet, which is generated using a MAC generation function.

Figure 8 shows the flow chart of the sIP layer. When an sIP layer intercepts a regular IP packet at the LOCAL_OUT hook of Netfilter, it identifies the packet as a secure packet if the destination address is a Class E address. (We

Table 2
Delays of Forwarding Components (in clock cycles)

	Authenticator Initialization	MAC Check	Secure Packet Translation	MAC Generation	Total	Effective Bandwidth
sIP	3067	-	-	3812	6879 (3.44 μs)	291 MB
SG	-	4463	450	3587	8500 (3.40 μs)	329 MB
SC	-	4455	-	3869	8324 (3.33 μs)	337 MB

choose Class E addresses as SHs in our testing for outgoing packets.) If it is, the sIP layer translates the packet into a host-secure packet. In the translation, the sIP layer first allocates 32-byte space for an authenticator from the tail room of an *sk_buff* using Linux *sk_buff* management function *skb_put()*. Then the sIP layer copies the destination address of the IP packet into byte 17 to 20 in the newly allocated 32-byte authenticator. i.e., the destination zone identifier and the destination host identifier fields. The sIP layer also replaces the destination IP address of the packet with the IP address of an SG. The source zone identifier and source host identifier field are filled with the zone identifier and the host identifier. The sIP layer further copies the protocol field in the IP header into the protocol field of the authenticator and updates the protocol field of the IP header to 135. (We use 135 as the SNS protocol number in our testing for incoming secure packets.) A 32-bit random number is also added into the authenticator. After the sIP layer initializes the authenticator, it generates a MAC using *hmac-md5* (RFC2085) and its host key.

Using the time stamp counter (TSC) of Pentium CPUs to directly read CPU clock cycles, we are able to measure the delay at each step of our implementation in clock cycles. We first measured the delay of four MAC generation functions using public-available codes from NIST, OpenSSL and IETF. As shown in Table 1, Hmac-md5 performs the best in both delay and memory. It takes 3812 clock cycles (1.91μs) to generate a MAC for a 24-byte SNS authenticator, and it requires 1MB memory for holding 65536 keys. Meanwhile, if given preprocessed key schedules,Blowfish-cbc-mac and AES-cbc-mac take fewer cycles than hmac-md5 and hmac-sha1 in the MAC generation. However, Blowfish requires 1042 32-bit sub-keys and AES requires 44 32-bit sub-keys for each master key. To generate a key schedule takes 127014 clock cycles

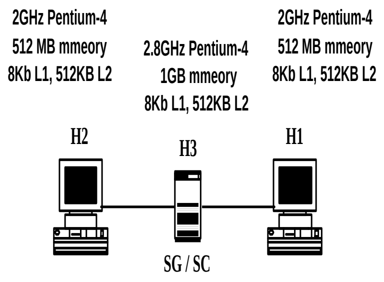
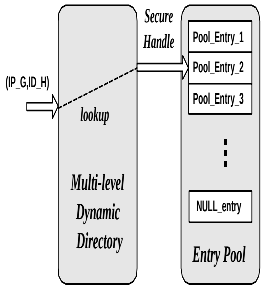
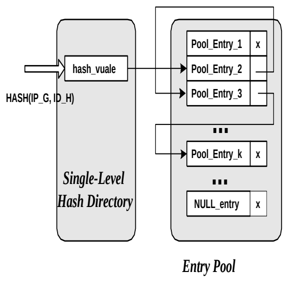


Fig. 9. Experimental Setting



(a) Multi-Level.



(b) Single-Level.

Fig. 10. Two Types of Organization of Remote Name Table.

for Blowfish and 9925 clock cycles for AES. These heavy costs make the two approaches impractical for the MAC generation. In addition, if we use preprocessed key schedules for Blowfish and AES, their memory requirements are rather high because we need maintain many keys on SGs and SCs. For example, if we need 65536 keys at an SG, using blowfish requires more than 133MB memory to store sub-keys, while using AES needs more than 11MB memory for sub-keys. Therefore, we choose hmac-md5 in the our implementation.

We present the additional delays caused by SNS components for authenticated packet forwarding in Table 2. We performed a stress test of sIP layer by sending a large number of UDP packets of 1024 bytes over a direct link between two hosts. In the first row, we show the measured average delays of authenticator initialization and MAC generation for a packet at a source host. The average delay of a packet caused by the sIP layer is 6879 cycles (3.44μs). In addition, we also measured the effect of sIP on end-to-end bandwidth using *Iperf* from NLANR (www.nlanr.net). On a 100Mbps dedicated link, we achieve a raw transmission rate of 93.9 Mbps over regular IP and a raw transmission rate of 91.9Mbps over sIP, 98% of the rate using IP. Furthermore, we performed a similar stress test of an SG and measured the average delays of packet authentication, secure packet translation, and MAC generation, as shown in the second row of Table 2. We connect host H1 to H2 through H3, which acts as an SG, as shown in Figure 9. The average delay of a packet caused by an SG is about 3.40 μs. Moreover, we performed a similar test in which H3 acts as an SC. The results are also shown in the third row of Table 2.

The average delay of a packet caused by an SC is about 3.33 μs. In summary, in a complete forwarding path from a source host to a destination host, the total extra delay caused by SNS forwarding components (sIPs, SGs and SCs) is about 20.34 μs. In the meantime, the last column in Table 2 also shows that our prototype can support a forwarding rate around 300 MBps, which is sufficient for a LAN environment with a 100Mbps or 1Gbps link. These experimental measurements on our prototype implementation have shown the feasibility of constructing SNS using regular PCs for common LANs.

5 Dynamic Table Management at an SG

The SH lookup mechanism of an SG is critical to its performance and scalability because it needs to lookup an SH for each packet from a potential large address table. Therefore, we focus on this issue in this section.

5.1 Table Operations and Requirements

In the process of secure name 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 pair of (IP_G, ID_H) and an SH to access an address entry. For a packet from a remote domain, we need to use its (IP_G, ID_H) as a primary key to find (or insert) its forwarding information into the address table, and then return an SH as the source IP address of a secure packet. As a result, when a local host sends a packet back to the remote host, it uses this SH as the destination address. When this return packet arrives at an SG, the SG directly accesses the corresponding address entry based on the SH. As a result, we are able to hide the (IP_G, ID_H) pair from a local host. Because we need to use both a primary key and an SH to access the same dynamic table, we cannot directly apply existing dynamic table management schemes such as linear hashing.

Therefore, we design a two-layer data 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 an insertion, we use a primary key to insert an address entry in the address table and return an index of the entry pool as a direct access handle (i.e., SH). For a lookup, we can either search the table based on a pair of (IP_G, ID_H) or directly access an address entry using an SH.

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 10. The multi-level directory scheme is shown in Figure 10.a, where each directory entry is corresponding to a unique address table entry. The single-level directory scheme is shown in Figure 10.b, where each directory entry is corresponding to a linked-list of table entries. In this scheme we need to compare primary keys to check if an entry is on the list.

The basic operations on the address table are insertion, lookup, and deletion. We focus on fast insertion/lookups in this paper. We have designed an efficient scheme to delete/recycle expired entries, which is presented in [14], due to the space limit of this paper. We have two types of insertions. In an *SNS Insertion*, a local host queries an SNS server for the address of a remote host. The SNS server performs a secure name resolution for this query, passes the (IP_G, ID_H) pair to a local SG, demands the SG to insert an address entry into the table for the (IP_G, ID_H) pair, and returns an SH to the local host, such that packets from the host will be forwarded correctly. In a *Routing Insertion* for an incoming connection, an SG inserts an address entry into the table and translates the incoming (IP_G, ID_H) pair into an SH, for setting up a correct reverse forwarding path. We also have two types of lookups. In an *SH-Direct*

Lookup, an SG needs to translate a packet from a local zone into a packet to a remote zone, and it uses the SH carried in the packet header to directly retrieve an address entry. In a *Routing Lookup* for an existing incoming connection, an SG already has a table entry with a corresponding SH. When a packet from the same remote host with the same (IP_G, ID_H) pair arrives, the SG finds the existing SH corresponding to the (IP_G, ID_H) pair, and uses this SH for packet forwarding between a local host and itself.

5.2 Dynamic Directory Schemes and Their Performance

We design two dynamic-directory schemes to achieve fast lookups and insertions. 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. Each directory entry consists of a flag F and a 32-bit pointer. $F = 0$ means that the directory entry is empty. While $F = 1$ means that the first 16 bits of a key is unique in the table, and the pointer field contains an SH, a direct index of the address pool. $F = 2$ means that multiple keys have the same first 16 bits, and the pointer field is refer to a sub-directory.

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 using C in Linux kernel. We extend the standard linear hashing scheme as a directory scheme to look up SH's. We start with a directory with 2^{16} entries, and then double the directory size as the table population grows.

```
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. 11. Lookup Algorithm of Multi-Level Directory

```
1. if $H_i$(key) $\geq p$
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. 12. Lookup Algorithm using Linear Hashing.

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 11 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 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.

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

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.

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

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.

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

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:

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

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 $M = \sum_{i=1}^{N_1} m_i$.

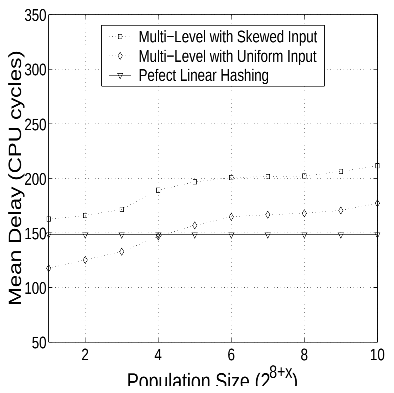
We now analyze the performance of the linear hashing directory scheme. Assume we initialize the directory with $\tilde{N_0}$ entries, say $\tilde{N_0} = 2^8$. Assume we have a perfect hashing function, then the memory cost of the single-level directory for a population of N_1 is denoted as $M_{N_1} = \tilde{N_0} \cdot 2^k$, 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 12.

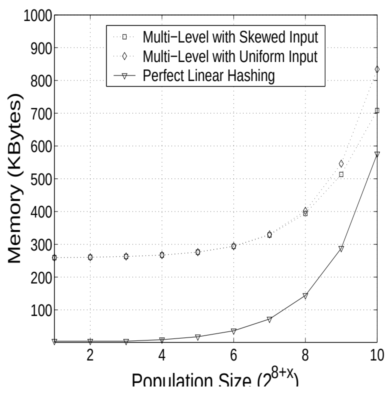
$$D_{lookup} = d_{hash} + d_p + D_{list} \tag{5}$$

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

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

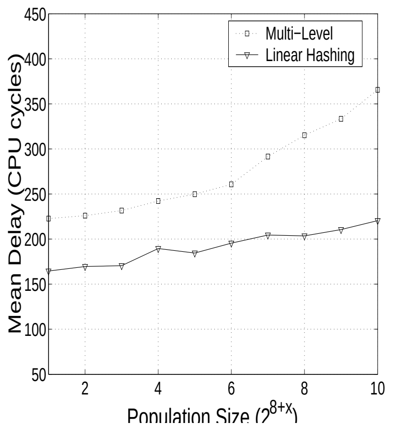


(a) Mean Delay.

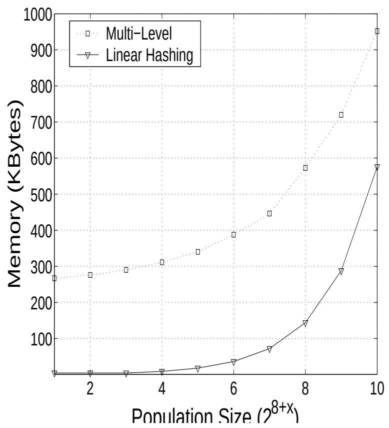


(b) Memory Cost.

Fig. 13. Comparison of Delay and Memory Cost with Models.



(a) Mean Delay.



(b) Memory Cost.

Fig. 14. Comparison of Delay and Memory Cost using Simulations.

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. Fig.13 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 Fig.13.b. 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. Fig.14.a shows the mean delay of the hashing scheme is significantly better than the multi-level scheme. Fig.14.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 and dynamic name binding, we can build a distributed filtering scheme to enforce packet authentication. We have described the basic design of the SNS framework, the SNS naming schemes and the SNS authenticated packet forwarding. We have further addressed the performance and scalability issues in the 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 fast secure-handle schemes to address the scalability issue in fast address translation.

To fully exploit the advantages of SNS, we still face several challenges such as further improving service availability and scalability. For example, we need to further protect SCs from attacks (such as packet replay and flooding) because they are exposed to attackers. We will address this issue from two perspectives. First, we will build a fast filter scheme on these SCs using Bloom Filter [17] in order to stop attacking packets into SNS domains. The main tradeoff of this scheme is between computation costs and probabilities that invalid packets penetrate the filter. Furthermore, we will investigate the effect of reconstructing dynamic packet forwarding paths to defeat attacks through resources within SNS domains and potential resources from a third party between SNS domains. 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 Security 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," *Technical Report, EE, Univ of Hawaii*, Oct., 2003.

[15] C. Choi, Y. Dong, and Z.-L. Zhang, "Implementation of SNS authenticated packet forwarding mechanism," *Technical Report, CS, Univ. of Minnesota*, Nov., 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.*