Effectively Fighting Common Spoofed and Unsolicited Packets via Lightweight Permits ⋆

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

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

[b] *Dept. of Computer Science, University of Minnesota, Minneapolis, MN 55455.*

Abstract

One of key security issues on the current Internet is unwanted traffic, the forerunner of unauthorized accesses, scans, and attacks. It is vitally important but extremely challenging to fight such unwanted traffic. We need a series of defensive mechanisms to identify unwanted packets, filter them out, and further defeat their associated attacks. In this paper, we propose a *lightweight, scalable* packet authentication mechanism, named *Lightweight Internet Permit System (LIPS)*, as a first line of defense to effectively filter out the most common forms of unwanted traffic, spoofed and unsolicited packets, such that in-depth security schemes can take care of the remaining issues more efficiently. LIPS is a simple extension of IP, in which each packet carries an *access permit* issued by its destination host or gateway, and the destination verifies the access permit to determine to accept or drop the packet. LIPS provides *preliminary traffic-origin accountability* that supports two salient features to confine unwanted traffic: 1) filter out the most common forms of unwanted packets and defeat associated attacks; 2) help us identify compromised hosts/domains such that we are able to build active defense schemes to deal with various attacks through real-time inter-domain collaboration. In this paper, we first present the design and prototype implementation of LIPS on Linux 2.4 kernel, and then use analysis, simulations, and experiments to demonstrate the efficacy of LIPS in protecting critical resources with light overheads.

Key words: network security, unwanted traffic, IP spoofing, packet authentication.

⋆ This work was supported in part by the National Science Foundation under the grants ANI-0073819, ITR-0085824, CNS 0435444, 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 NSF.

* Corresponding author, TEL:01-808-956-3448 FAX:01-808-956-3427.
Email address: yingfei@hawaii.edu (Yingfei Dong).

1 Introduction

Implicit in the original design of the Internet is *open trust* [1] – that end hosts/end users are always trusted – with its associated *lack of accountability*. Such an open trust model allows malicious users to exploit vulnerabilities of networks and applications to launch various cyber attacks, while enjoying the comfort of not being tracked down easily. In particular, on the current Internet, a source IP address can be easily spoofed and manipulated, and *unwanted* packets can intrude an unwary host with ease (despite firewalls), which is often tricked into unintentional "accomplice" (e.g., in the case of viruses and worms), amplifying and spreading attacks to many other vulnerable hosts. To combat this problem, many organizations choose to close off their networks via mechanisms such as VPNs [2–4], or employ firewalls to block certain types of packets (e.g., based on IP addresses, ports, or packet payload), regardless of senders and their intents. Clearly, such solutions are fairly limited in their scope or effectiveness as worms and email viruses can routinely penetrate firewalls. Furthermore, they are rather *rigid*, sometimes breaking existing applications and potentially impeding creation and deployment of new services and applications. There is still much debate in the networking research community regarding how to secure and fortify the current Internet while without jeopardizing its open architecture and end-to-end design principle [1].

As unwanted traffic becomes one of key security issues on the Internet, stopping unwanted traffic is vitally important but extremely challenging. We need to construct a series of security schemes to identify unwanted packets, filter them out, and further defeat their associated attacks. In this paper, we propose a novel *Lightweight Internet Permit System (LIPS)* to provide preliminary traffic accountability through simple, fast packet authentication for getting rid of most unwanted packets. By unwanted packets, we mean packets *not* intended for normal communications between hosts, such as packets with spoofed IP addresses, or generated by compromised hosts for port scanning or service probing. Such packets account for, or are forerunners of, most of unauthorized accesses, intrusion, disruption, DoS attacks and other cyber threats in today's Internet. For example, in analyzing the netflow data collected at the University of Minnesota's Internet border gateway, we have found that a disproportionally large percentage of the flows consist of one or a few packets, with no corresponding responsive flows in the reverse direction, and thus can be deemed as "unwanted". More in-depth investigation reveals that a majority of these flows are port scanning or known attacks that are blocked by firewalls and patched hosts.

LIPS is designed as a fast, scalable traffic authentication mechanism to filter out most common illegitimate packets, with minor changes to current systems and negligible overheads. We implement LIPS as a small patch to the IP layer at a host or gateway. When a source wants to communicate with a destination,

it first requests and obtains (if granted) an *access permit* from the destination. It then inserts a destination access permit into each packet sent to the destination. Only packets with proper access permits will be accepted at the destination. This simple architecture provides a scalable and flexible framework for establishing *preliminary traffic accountability* among networks and hosts, and for securing Internet resources *without* sacrificing their open and dynamic nature. Furthermore, LIPS also simplifies and facilitates the early detection of, and timely protection from, network intrusion and attacks by requiring valid access permits before any data packets can be accepted and processed. Hence by incorporating *active monitoring* and *rapid response* mechanisms with LIPS, we can build an effective and scalable *first-line* defense to protect Internet resources from unwanted traffic. However, as a weak form of authentication scheme, LIPS has its own limitations. First, it focuses on confining randomly spoofing and probing traffic; for other security requirements such as confidentiality, integrity, and strong authentication, it must be combined with in-depth security schemes. Furthermore, its lightweight nature determines that it is able to address most common security threats such spoofing and random probing, but not advanced attacks such as active replay attacks, as discussed at the end of Section 3. It depends on other in-depth security schemes to completely defeat such active attacks, although in Section 4 we have shown that such active replay attacks may only cause rather limited damages.

Related Works. Many security mechanisms have been proposed to defeat unwanted traffic, IP spoofing, and trace back attacking sources. Firewalls are broadly deployed with packet filters based on statically configured known signatures. They are unable to response to emerging attacks and can not help in automatic defense and inter-domain collaboration. Ingress filtering allows an ISP to prevent attackers from spoofing IP addresses outside the address space of a stub domain. It is effective for stub domains but not *transit* domains [5]. It also does not preclude an attacker spoofing within a legitimate address range. Since an ISP does not directly benefit from ingress filtering, an ISP has less incentives to deploy it. Spoofing Prevention Method (SPM) [6] enhances ingress filtering by providing better deployment incentives for ISPs, since it provides better protection for SPM-aware domains. A packet exchanged between two SPM domains is attached with a domain-specific temporal key, and the key is verified at the destination domain. However, similar to ingress filtering, SPM can not stop spoofing in a legitimate address space and does not help trace back to compromised hosts. IPv6/IPsec Authentication Header (AH) and associated VPNs establish shared keys to secure end-to-end communications in an all-or-nothing fashion. Due to the rigid requirement of pre-existing credentials, they are difficult to deploy. In the meantime, they all bear heavy overheads [7,8] in Security Association (SA) management, heavy cryptographic check, and maintaining per-flow states. Similarly, the visa protocols [9] use encryption and data signatures to authenticate a flow of pack-

ets. They also require shared keys established between access-control servers on a *per-source-destination* basis. Newly formed IETF Better-Than-Nothing-Security (btns) [10] provides unauthenticated keying for IPsec to create SAs between peers who do not possess pre-existing authentication credentials, e.g., self-signed certificates or bare public keys. It aims to address the deployment issue of IPsec. It is similar to LIPS in not requiring pre-existing credentials. In addition, Host Identity Protocol (HIP) [11] and Statistically-Unique-and-Cryptographically-Verifiable (SUCV) identifiers [12] focus on the address ownership problem by using cryptographic name spaces to address the spoofing issue.

Moreover, several approaches have been proposed to modify intermediate routers to defeat DoS attacks and IP-spoofing. Pushback [13] treats DDoS as a congestion-control problem and requires each router to detect and preferentially drop packets that probably belong to an attack. Upstream routers are also notified to drop those packets such that the routers resources are used to route legitimate traffic. The network capability scheme [14] inserts special tokens into packets and requires intermediate routers to check these tokens along forwarding paths for restricting unwanted packets. While these routers authenticate packets and maintain *per-flow states*, destination hosts also keep *per-flow states* for authentication using hash chains. Hop-integrity protocols [15] provide secure communications between adjacent routers by computing a message digest for *each packet* at *each forwarding step*. IP Easy-Pass [16] aims to protect real-time priority traffic from Denial-of-QoS attacks at an ISP edge router by maintaining *per-flow states*. SAVE [17] propagates valid source addresses between intermediate routers on forwarding paths. Several traceback schemes (e.g., IP traceback[18]) are proposed to track down attacking sources. Since these schemes require to change intermediate routers or maintain per-flow states, they are difficult to deploy. Furthermore, overlay approaches (SOS and Mayday [19,20]) exploit a *wide-area* overlay infrastructure with a *large number* of intermediate nodes to hide critical servers and filter out attacking traffic. Secure-i3 [21] also use an overlay network to hide the IP addresses of end hosts and give them the ability to defend against attacks by dynamically removing private triggers to stop unwanted flooding traffic. However, the data transmission delay in Secure-i3 is relatively high. Client puzzles [22] introduces a cryptographic-based challenge scheme against connection depletion attacks, such as TCP SYN attacks. It increases the difficulty for attackers but can not totally stop attacks.

The recently proposed Active Internet Traffic Filtering (AITF) [23] has a similar goal as LIPS. It leverages the routes recorded on incoming packets to identify the last point of trust on each attack path, and block attack traffic at that point, with the help of collaborative routers across multiple intermediate domains. Once the victim detects attack flows, it asks its gateway to block the flows; the gateway then requests the farthest gateway in the route record to

block all packets of these flows. If the gateway does not responds, the victim gateway escalates the filtering request to the second farthest gateway in the route record to block all traffic from the farthest gateway to the victim domain. The escalation continues until a gateway along the attack path responds. The key advantage of AITF is distributed filtering along the forwarding paths, under the assumption that AITF is deployed in a significant portion of the Internet. We will further compare AITF with LISP at the end of Section 4.

In summary, these approaches generally incur high computational overheads, or heavy key management costs, or require modification of intermediate routers, or broad infrastructure support. Therefore, they usually significantly degrade end-to-end performance for security, and are difficult to deploy. On the contrary, LIPS does not require pre-existing shared secrets, i.e., no key management costs; LISP uses secure hash instead of encryption or digital signatures, hence the overheads of cryptographic check are significantly reduced; LIPS is an end-to-end/edge-to-edge approach that does not require changes of intermediate networks. Although LIPS is subject to two active attacks: payload replacing and permit replay, as discussed later in this paper, the cost of successful attacks is extremely high and the damage is fairly limited. Furthermore, these attacks can be easily taken care of by in-depth security schemes commonly used in applications such as SSL.

The remainder of this paper is organized as follows. In Section 2, we present the basic concepts and constructs of LIPS, illustrate how it works and why it is useful. In Section 3, we present the design and implementation of LIPS. In Section 4, we evaluate the performance and scalability of LIPS through simulations and experiments on our prototype implementation. We conclude this paper in Section 5.

2 Basics of LIPS Architecture

The idea of LIPS is simple: every LIPS packet carries an *access permit* issued by its destination, and this permit is verified at the destination to determine whether the packet is accepted or dropped. Hence for a source, it must first obtain a valid destination access permit. This simple mechanism enables the destination to easily *eliminate illegitimate/spoofed packets* and *control who has access to it*, e.g., through a simple security policy database. Only data packets with valid access permits will be accepted and passed to applications on a destination host. Thus a malicious host cannot simply inject unwanted traffic to harm a destination host without first requesting an access permit and identifying itself to the destination. In the following, we first introduce the basic constructs and operations of LIPS, and illustrate how access permits are generated, exchanged, and verified. We then discuss the advantages and limitations of LIPS at the end of this section.

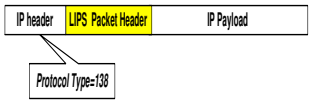


Fig. 1. Converting an IP packet into a LIPS packet.

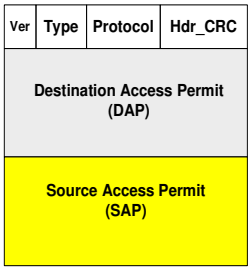


Fig. 2. Format of LIPS Packet Header.

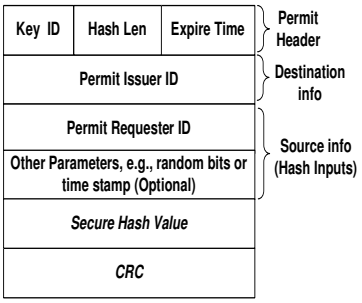


Fig. 3. Format of LIPS Permit.

LIPS Packet. LIPS is a simple extension of the IP protocol. We convert an IP packet into a LIPS packet by inserting a *LIPS header* into the payload field of the IP packet and changing the protocol type to 138 in the IP header, as shown in Fig.1. To avoid the potential segmentation issue, we first perform a path MTU discovery and then set a proper MTU for the connection. We choose 138 as the protocol type of LIPS in our prototype implementation. The format of a LIPS packet header is given in Fig.2, which includes four control fields, a *destination access permit (DAP)*, and a *source access permit (SAP)*. A DAP is issued by a destination to a source. It is carried in packets from the source to the destination and verified at the destination. A SAP is issued by the source to the destination for packets on the return path. The *Ver* field holds a LIPS version number. The *Type* field specifies the type of a LIPS packet, such as a permit request, a permit reply, or a LIPS data packet. Since we replace the IP protocol type in the IP header to 138 when we translate an IP packet into a LIPS packet, we use the *Protocol* field to hold the protocol type of an original IP packet such that we can restore the original protocol type after the LIPS packet is accepted at a destination. The *Hdr_CRC* field is a simple CRC for a LIPS packet header.

Access Permit. An access permit is constructed using *keyed* message authentication code (MAC) [24] at a LIPS-aware host. This MAC is generated through a secure hash function with two inputs: a *plain* hash message (chosen by a permit issuer and carried in an access permit in plain text) and a *secret* hash key held by the issuer. For example, we can simply use the IP address of a permit requester as the hash message.

As shown in Fig.3, an access permit includes five main fields: a *permit header*, a *permit issuer's ID*, a *permit requester's ID* (plus optional parameters), a *secure hash value*, and a *CRC checksum* of the secure hash value. The permit header contains an index (*Key ID*) of a secret key used for this permit at its issuer, a hash length (*Hash Len*) that specifies the length of the secure hash value in this permit, and a permit expire time (*Expire Time*) that defines the effective duration of this permit. The length of secure hash value can be adjusted from 64 bits to 128 bits depending on the permit issuer's security requirements. The source information (denoted as M) includes, e.g., the source IP address

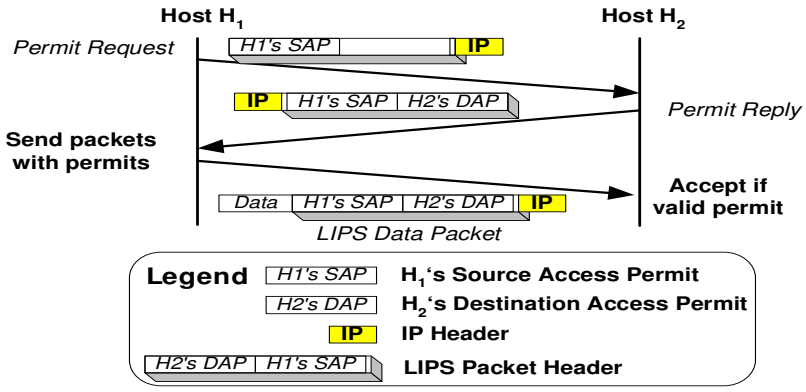


Fig. 4. LIPS Message Exchanges for Setting up Access Permit.

and several optional security parameters. For example, an optional sequence number can be used to deal with permit-replay attacks, which is activated only when a replay attack is detected. Therefore, in the normal operations, no per-connection states are maintained at a permit issuer. The source information is used by the issuer as the input plain hash message to a secure hash function to compute a message digest, $H(M, K_t)$, where $H()$ is a secure hash function, e.g., HMAC-MD5 [25], and K_t is a secret key of the permit issuer at time t. For ease of exposition, we use the source IP address as M in the following presentation. Given that the hash length in the permit header is l, the secure hash value of a permit is the first l bits of a message digest. For example, we can choose the first 64 bits of a 128-bit HMAC-MD5 digest as a secure hash value. This hash value will be used for validating the permit. Note that the hash value is *specific* to the requester, and is valid only for a short period of time, as K_t will be changed over time. Without knowing the secret key, it is very difficult to forge a permit. The idea of using secure hash functions to avoid storing states in LIPS is similar to the idea of SYN cookies [26]. However, a LIPS permit works in the network layer for all protocols, while a SYN cookie works in the transport layer for TCP only. Furthermore, a permit is highly aggregated for all traffic to a domain/host, while a SYN cookie is for only a TCP connection.

Exchange of Access Permits between a Source and a Destination. We use a simple example to illustrate how access permits are set up between two LIPS-aware hosts. As shown in Fig.4, when a source host H_1 wants to communicate with a destination host H_2, H_1 first sends a *permit request* to H_2, which carries H_1's SAP. The SAP contains a secure hash value generated based on a *secret* hash key of H_1 and a plain hash message about H_2 (e.g., H_2's IP address). *Note that not all hosts will be allowed to access* H_2. A *security policy* at H_2 is checked to determine if H_2 accepts this permit request. If it does, it generates an access permit (H_2's DAP for H_1), containing a secure hash value generated based on a secret hash key of H_2 and a plain hash message about H_1 (e.g., H_1's IP address). Then H_2 sends the permit (as the SAP) in a *permit reply* message back to H_1, using H_1's SAP (attached in the permit

Destination IP address	Flag	Destination Access Permit
172.10.10.1	1	xxxxxxxx
129.128.128.1	3	NULL

Fig. 5. Permit Cache Examples.

request from H_1) as the DAP of the reply. This simple example shows the case when a LIPS-aware host directly exchanges permits with another in a LIPS host mode introduced in Section 3.1. We will further introduce domain-level permit exchanges in Section 3.2 when we present the LIPS gateway mode.

H_1 will only accept a permit reply that carries a valid DAP, namely, a SAP issued by H_1 to H_2 in an earlier permit request. This is done by computing a secure hash value using the plain hash message carried in the DAP and a secret key pointed by the key index. If this hash value matches the secure hash value carried in the DAP, H_1 accepts this reply and caches the SAP of the packet (i.e., H_2's DAP) into a *permit cache*. For the subsequent data packets sent to H_2, H_1 puts both H_2's DAP and its own SAP into the LIPS headers of these packets. When H_2 receives a LIPS packet from H_1, it verifies the DAP of the packet and accepts it only if the DAP is valid.

Security Policy. For a common host, a sample policy may be only accepting permit requests from a local domain, including other local hosts and its security gateway(s). As a result, such a host only accepts packets from other local hosts, or its security gateway(s) when communicating with hosts in other domains. As we introduced later in the LIPS gateway mode, remote hosts have to go through the permit server of this host to gain accesses to it. This naturally stops random scanning and worm spreading packets from other domains, i.e., an attacker outside the local domain can not directly discover vulnerabilities via scanning, and a worm can not easily propagate across domains through random probing. Consequently, potential damages are confined and localized. For a server, a default policy is to accept permit requests from a set of known domains (or all domains in the case of a web server). We design LIPS as the mechanism to enforce chosen policies. We do not explore the details of policy management as it would be a whole set of different issues involved and is needed to be addressed separately. We refer interested readers to the KeyNote Trust-Management System [27].

Permit Cache. Each LIPS host maintains a permit cache with a format shown as Fig.5. A cache entry contains a destination IP address, a *Flag*, and a destination access permit. For incremental deployment, the cache not only holds permits for LIPS-aware hosts, but also tracks non-LIPS hosts (if allowed by security policies), using the destination IP address as its primary index. The flag is used to distinguish the state of a cache entry: flag = 0, indicating that the entry is in initialization, namely, a permit request has been sent to the destination but the reply has not been received yet; flag = 1, a valid permit

for the destination is available; flag = 2, the destination permit has expired; and flag = 3, the destination does not supports LIPS.

Key Management. LIPS uses an extremely simple key management scheme: *each host keeps its own keys and no key exchanges are required.* Each LIPS host maintains a secret key pool of, say, 256 keys. Each key is uniquely identified by a key index. When a host generates an access permit, it randomly chooses a key from its key pool and records the key index in the *Key ID* field of a permit header. When a host verifies an access permit, it retrieves a key using the *Key ID* of the permit header. Note that in LIPS a key pool is *not shared* with any other hosts, and *no key exchange* among hosts is required, contrary to the complex key establishment procedures in other approaches. Although in the LIPS gateway mode (see section 3.2) permit servers and security gateways do share a secret key pool, conceptually they are two facets (*control* vs. *data*) of the same security unit, and often can be implemented as a single box. Hence the overhead of key management in LIPS is minimal.

Why LIPS. LIPS not only helps us to confine IP-spoofing, random probing, and associated attacks, but also assists us to identify attacking sources such that we can fix them through local and inter-domain collaboration. First, LIPS supports simple traffic-origin accountability that allows destination domains and hosts to deny illegitimate accesses and stop common unwanted traffic such as IP-spoofed and random probing packets and associated attacks. This gives ISPs *strong incentives* to deploy LIPS. Since packets without valid permits are automatically discarded, LIPS naturally filters out random reflection and probing packets (used in reflection attacks and worm spreading, or for attackers to collect network information). Scanning attacks (such as worms) generally rely on port scanning to find vulnerable hosts, tricking them to execute the malcode carried in the payload, and thereby compromising them as stepping stones or unintentional accomplices to further spread attacks. As discussed earlier, a simple LIPS security policy at a host can help us restrict random scanning packets and therefore defeat associated attacks.

More importantly, LIPS facilitates and simplifies the tasks of detecting unauthorized intrusion and attacks by forcing malicious hosts to first request access permits and identify themselves to intended targets before launching attacks. As a result, we can identify attacking sources, i.e., compromised hosts or domains, and deploy real-time defense schemes that automatically fix these sources through local measures and inter-domain collaboration. Since a malicious host cannot simply inject unwanted traffic to harm a LIPS-protected destination without first requesting an access permit, we can better defend against such attacks by simply detecting anomalies in the permit request traffic. For example, a sudden unusual surge of permit requests to one or more hosts signifies suspicious activities. In particular, when implemented in the gateway mode (as in Section 3.2), a source zone can detect attacks originating

from malicious hosts within its zone and quarantine them by denying their permit requests. Hence combined with network intrusion mechanisms, LIPS can form an effective first line of defense against cyber attacks by stopping most common unwanted traffic.

In addition, LIPS helps us to mitigate request-based DoS attacks, which trick target hosts to expend their valuable resources by unnecessarily processing bogus service requests. LIPS also helps us to reduce the damages of flood-based DoS attacks, which intend to overwhelm a target host or link, since flooding packets without valid permits are simply dropped without reaching the target, and therefore cannot harm applications on targets, as shown in Section 4. Moreover, as a domain-to-domain approach, LIPS is *incrementally deployable* since it does not require changes in backbone networks as many other approaches, and it only needs minor software patches on common platforms. It also largely reduces the load of IDSs by filtering out most unwanted packets and allowing IDSs to focus on serious threats.

3 LIPS Design and Implementation

For incremental deployment and scalability, we design LIPS operating in two modes. The basic LIPS works in a *host mode*, in which a LIPS-aware host directly communicates with another LIPS-aware host as illustrated in Section 2. To improve its security strength and capability, we further develop the LIPS *gateway mode*: We organize LIPS-aware hosts into *secure zones* based on their network administrative domains or zones. For intra-zone traffic, hosts communicate with each other in the host mode; for inter-zone traffic, we use *permit servers (PSs)* to manage inter-zone permits and employ *security gateways (SGs)* to verify inter-zone packets. The LIPS gateway mode can be easily deployed in a large scale without changes of intermediate domains.

3.1 LIPS Host Mode

The LIPS host mode is used as an incremental approach to deploy LIPS when LIPS-aware hosts directly communicate with each other in a small scale. Eventually, when LIPS is adopted by many domains in a large scale, the host mode will be used for intra-zone communications under the gateway mode. To support the host mode, we install a *Host Authentication Layer (HAL)* at a LIPS-aware host, which is a small patch to the IP layer. The main functions of HAL include exchanging permits, maintaining a permit cache, attaching permits to packets, and verifying access permits, as explained in the following. We implemented HAL in Linux 2.4 kernel using *Netfilter*. The HAL uses 256

128-bit secret keys and employs HMAC-MD5 [25] as the secure hash function. As reported in Section 4, our *software* implementation on a common Linux platform can achieve a packet authentication rate of 643 Mbps, which shows the feasibility of LIPS on common hosts.

Packet Processing in HAL. The HAL intercepts each *outbound* packet in the *ip_output()* procedure of the IP layer, and looks up its permit cache based on the destination IP address of the packet. If a valid permit is found, it converts the IP packet into a LIPS packet, attaching a DAP (from the cache) and a SAP (its own access permit generated for the destination); if the HAL finds that the destination is non-LIPS host, it simply passes the packet back to the IP layer without changes or drops the packet depending on its security policy; if no entry is for this IP address in the cache, or we find an entry expired or in initialization, the HAL puts this packet into a permit waiting queue and initializes a permit setup procedure introduced in the following.

The HAL also intercepts each *inbound* packet in the *ip_rev()* of the IP layer. If it is a LIPS data packet, the HAL checks the DAP of the packet. If valid, the packet is accepted and its SAP is refreshed in the permit cache based on the source IP address. If it is not a LIPS packet, the HAL simply passes it to the IP layer or drops it. If it is a LIPS permit request/reply, the HAL executes the permit setup protocol as follows.

Permit Setup Protocol. To obtain a permit of a destination, the HAL at a source sends a permit request to the destination, creates/updates a cache entry with a flag of 0 (i.e., in setup), and sets a retransmission timer for the request. Upon receiving this request, the HAL at the destination generates a permit and sends a permit reply to the requester as introduced in Section 2. When the HAL at the requester receives this reply, it first verifies the DAP of the reply. If the DAP is valid, it searches through its permit waiting queue and processes all packets destined to this destination. Then it stores this permit into its permit cache and sets the status flag of the corresponding entry to 1 (i.e., a valid permit).

In case a permit request or reply is lost, a HAL may mistake a LIPS-ware host as a non-LIPS host and may reject its accesses. We solve this issue by setting an effective period for each cache entry. The HAL will activate the permit setup protocol again once this entry is expired, such that it will eventually obtain an access permit from a LIPS-aware destination and gain accesses [1] .

[1] For incremental deployable, we may allow a LIPS-aware *source* host to initialize a communication with a non-LIPS *destination* host (but not vice versa!). In this case, the HAL at a permit requester will: 1) receive an ICMP protocol unreachable message from the destination, and set the status flag of the corresponding entry to 3 (i.e., a non-LIPS host). 2) see a permit retransmission timeout due to no responses. At the first two timeouts, the HAL resends the permit request to the destination. If

Permit Lookup/Insertion. We have introduced the basics of a permit cache in the previous section. For each LIPS data packet, we need to find a destination access permit at the sender's permit cache and refresh/insert a source access permit at the receiver's permit cache. Therefore, we must minimize this lookup delay. Furthermore, we must carefully use the kernel memory for a permit cache. To shorten lookup delays and minimize memory costs, we organize a permit cache using a linear hashing scheme with controlled-splitting [28], which not only has $O(1)$ expected lookup/insertion delay, but also is extremely memory-efficient, as its table size linearly grows with its population. We refer interested readers to [29] for the details of our permit lookup schemes. We evaluate its performance in the next section.

The LIPS host mode has two limitations. First, it can not prevent a flooding attack from directly hitting a LIPS host. Furthermore, it is not scalable because maintaining a large number of host-specific permits will degrade the performance of LIPS host-mode.

3.2 Design of LIPS Gateway Mode

To improve the scalability and security strength of LIPS host mode, we develop the LIPS gateway mode, which employs a two-tiered trust model: LIPS-aware hosts are organized into secure zones based on their network administrative domains. We use *zone access permits* to authenticate *inter-zone* packets, and use *host access permits* (as in the host mode) to authenticate *intra-zone* packets. Each zone has a *permit server (PS)* to manage inter-zone permits and a *security gateway (SG)* to validate inter-zone packets based on inter-zone permits. Once an inter-zone permit is established between a pair of zones, the subsequent communications between them will take advantage of this permit and avoid repeatedly setting up inter-zone permits, i.e., we only need to establish one inter-zone permit for all communications between them. As a result, we not only reduce permit setup delays but also significantly reduce inter-zone permit exchange traffic. Furthermore, we propose a unique and simple *permit-mutation* method to transform zone permits and host permits back and forth such that *not only security gateways do not need to keep per-flow states but also zone permits are not revealed to hosts*. Since it is rather difficult to gain accesses to intermediate routers and links, attackers have very little chances to sniff zone permits. Therefore, permit-mutation *localizes damage* within a zone. We will evaluate this limited damage in the next section. Within each zone, LIPS-aware hosts still directly communicate with each other using the LIPS host mode.

Permit Server, Intra-zone and Inter-zone Permit Setup Protocol. As show in Fig.6, host H_1 in zone Z_1 wants to access host H_2 (e.g., a protected

still no responses at the third timeout, the HAL treats the destination as a non-LIPS host.

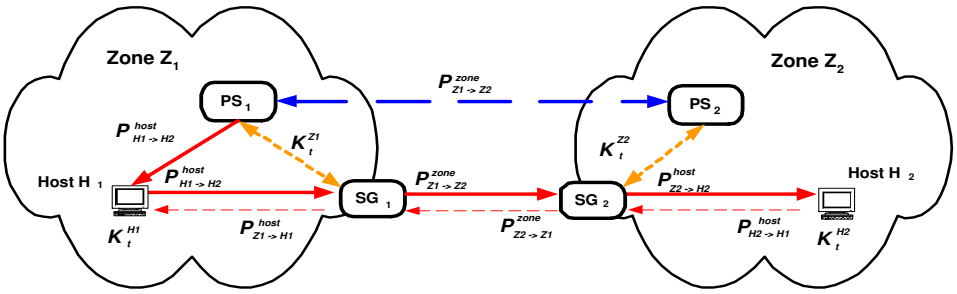


Fig. 6. Illustration of LIPS Gateway Mode.

Table 1. Summary of Notations.

Notation	Definition
$P^{host}_{x \to y}$	Host Access Permit issued to host x by host y
$P^{zone}_{x \to y}$	Zone Access Permit issued to zone x by zone y
IP_x	IP address of host x
$\oplus$	Bitwise Exclusive OR
PS_i	Permit Server of Zone i
SG_i	Security Gateway of Zone i
$K_t^{H_x}$	Secret hash key of Host H_x at time t
$K_t^{Z_i}$	Secret hash key of Zone Z_i at time t

application server) in zone Z_2. PS_1 is the permit server of zone Z_1, and PS_2 is the permit server of zone Z_2. Zone Z_1 and zone Z_2 are protected by security gateways SG_1 and SG_2, respectively, which authenticate both ingress and egress traffic originating from and destining to trusted hosts in these zones. To obtain a permit to access remote host H_2, H_1 authenticates itself to its local permit server PS_1 (e.g., via a local authentication scheme). PS_1 assists H_1 to obtain an access permit to H_2. Under the two-tiered model, we divide the packet forwarding path from a local host H_1 to a remote host H_2 into three segments: from H_1 to SG_1, from SG_1 to SG_2, and from SG_2 to H_2. Correspondingly, we use three access permits at each of these segments for packet authentication: an intra-zone host access permit $P^{host}_{H_1 \to H_2}$, an inter-zone permit $P^{zone}_{Z_1 \to Z_2}$, and another intra-zone host access permit $P^{host}_{Z_2 \to H_2}$. We introduce the setup protocols for these permits in the following. The setup of permits for the reserve path from H_2 to H_1 is similar. Table 1 summarizes the key notations used this discussion.

Each PS is assigned a *zone ID*. In this prototype design, we simply choose the IP address of a PS as its zone ID since inter-zone permit requests and replies are exchanged between PSs. We use this zone ID to generate a zone access permit as follows. In response to a permit request from a trusted host, the local PS passes the request to the corresponding (authoritative) PS in the remote secure zone, together with its zone ID and other necessary credentials. (For a PS to find an authoritative PS of another domain, we add a simple resource record at the DNS of a domain such that a PS can find another PS

through a simple DNS query, based on a simple name convention. We assume that DNSsec will solve security issues related to current DNS and so we will not discuss DNS security in this paper.) If the access is allowed, the remote PS will generate a *zone access permit* (or *zone permit* in short) based on the local PS's zone ID. Hence the access permit is *source-zone specific*. The remote PS returns the zone permit to the local PS together with its *own* zone ID. Instead of directly passing the zone permit to the requesting host, the local PS creates a new *host access permit* (or *host permit* in short) by adding some "random" value generated based on the source and destination IP addresses as explained in the following. This *mutation* of a zone permit into a host permit makes the host permit *specific to both source host and destination host*, thereby rendering it difficult to be spoofed by other hosts or to other destinations.

Zone Access Permits. Zone access permits are generated in the same fashion as host access permits but use a zone ID as a plain hash message. For a packet, let use IP_1 to denote its source IP address of host H_1 in zone Z_1, and use IP_2 to denote its destination IP address of host H_2 in zone Z_2. Let IP_{PS_1} be the IP address of a requesting PS (as a zone ID), and $K_t^{Z_2}$ be a *secret key* maintained by the queried PS_2 at time t. Then the secure hash value of the zone permit is $P^{zone}_{Z_1 \to Z_2} = H(IP_{PS_1}, K_t^{Z_2})$, where $H()$ is a secure hash function, and the CRC checksum of the permit is computed on $P^{zone}_{Z_1 \to Z_2}$. For ease of exposition, we will also refer to the secure hash value contained in an access permit as simply the access permit. As explained in the following, the CRC checksum is used to verify the validity of the permit after the *permit de-mutation* for outbound packets. Note that the generated permit is *specific* to the requesting zone, and is valid only for a certain period of time, as $K_t^{Z_2}$ changes over time. Without knowing $K_t^{Z_2}$, it is very difficult to forge a zone permit.

Mutation of a Zone Permit to a Host Permit. Given the zone permit $P^{zone}_{Z_1 \to Z_2}$, the requesting PS *mutates* it into a host permit $P^{host}_{H_1 \to H_2}$ using the IP address of the requesting (source) host, IP_1, and the IP address of the queried (destination) host IP_2. Let $K_t^{Z_1}$ be a *secret key* maintained at the requesting PS at time t. We construct a host permit, $P^{host}_{H_1 \to H_2} = P^{zone}_{Z_1 \to Z_2} \oplus H(IP_1, IP_2, K_t^{Z_1})$. Note that the host permit $P^{host}_{H_1 \to H_2}$ is only valid for the source H_1 to access the destination H_2 for a certain period of time. Again, without knowing the secret key $K_t^{Z_1}$, it is also very difficult to forge a host permit. The host permit is essentially the same as the zone permit, with the secure hash value $P^{zone}_{Z_1 \to Z_2}$ replaced by $P^{host}_{H_1 \to H_2}$. Note that the CRC checksum is *not* re-computed.

Host and Gateway Operations. At both source and destination domains, we establish lightweight packet authentication mechanisms for verifying and filtering packets based on host and zone access permits.

Host Operations. In the gateway mode, we also install a *host authentication layer (HAL)* at each host as in the host mode. However, this HAL works a

little bit differently. During its initialization, a HAL authenticates itself to its permit server and security gateways via a local authentication scheme, e.g., Kerberos [30]. This authentication only occurs once during its initialization. In the meantime, it also issues *host access permits* to its PS and its SG for authenticating permit replies and LIPS data packets from them, e.g., host H_1 issues permit $P^{host}_{Z_1 \to H_1}$ to PS_1 and SG_1.

The HAL layer at an end host x intercepts each outbound packet and then looks up its permit cache based on the destination IP address of the packet. Similarly to the HAL in host mode, if a destination access permit is found, it is attached to the packet. In addition, the host will attach its source access permit generated using its local zone ID IP_{PS_i}, $P^{host}_{Z_i \to x} := H(IP_{PS_i}, K^{H_x}_t)$, where $K^{H_x}_t$ is a secret key kept by the host at time t. This source access permit is used for authenticating packets from the security gateway to the host. For each *incoming* packet, the HAL checks the validity of the destination permit using the *destination zone ID* (carried in the permit) and its own secret key. (It is the *reverse* operation of generating the source access permit in the above). The packet is accepted only if it passes the verification. In this case, the source access permit is *cached* in the permit cache (with a timer appropriately set, in a manner similar to the ARP table used for IP and MAC translation).

Gateway Operations. The *gateway authentication* layer (GAL) is a LIPS realization at a SG, which is a small patch to the IP layer. For *outgoing packets*, the SG is responsible for ensuring that they are authorized to access the protected remote zones and hosts. To verify this, it uses the source IP address IP_1, the destination IP address IP_2, and the destination access permit $P^{host}_{H_1 \to H_2}$ (carried in the packet) to first compute $X := P^{host}_{H_1 \to H_2} \oplus H(IP_1, IP_2, K^{Z_1}_t)$, where $K^{Z_1}_t$ is the secret key that the SG shares with the local PS. It then generates the checksum on X. If the computed checksum *does not* match the checksum carried in the destination access permit, the authentication fails and the packet is dropped. Otherwise, the secure hash value in the destination access permit is replaced by X (note that $X = P^{zone}_{Z_1 \to Z_2}$), and thus the destination access permit is *de-mutated* back to the original zone access permit issued by the destination zone. Furthermore, the (host) source access permit of H_1, together with the source host IP address, is cached in the LIPS permit cache at the gateway. In addition, the gateway will replace the (host) source access permit in the packet with a new (*zone*) source access permit, $P^{zone}_{Z_2 \to Z_1} := H(IP_{PS_2}, K^{Z_1}_t)$, where IP_{PS_2} is the IP address of PS_2 as the *destination* zone ID. $P^{zone}_{Z_2 \to Z_1}$ is used to authenticate packets from the destination zone Z_2 on the reverse path.

For packets entering a destination zone, the security gateway is responsible for verifying that they carry proper zone access permits. This is done by checking to see whether the destination permit carried in an incoming packet, $P^{zone}_{Z_1 \to Z_2}$, is valid. If this verification fails, the packet is discarded. Otherwise, the packet is allowed to enter the destination zone. Using the destination IP

address IP_2, the gateway looks up its permit cache and replaces the destination zone permit with the corresponding destination host permit. Depending on whether the destination host is a trusted host (e.g., a server) in a *protected* (e.g., secluded) network, or a client host in a less secure environment, the gateway may replace the source *zone* access permit, $P^{zone}_{Z_2 \to Z_1}$, with a mutated source *host* access permit, $P^{host}_{H_2 \to H_1} := P^{zone}_{Z_2 \to Z_1} \oplus H(IP_1, IP_2, K_t^{Z_2})$. In the former case, for scalability this operation is *optional* so that trusted servers and other high-performance hosts in protected networks only need to maintain zone-level access permits. In the latter case, this operation would prevent other untrusted hosts to eavesdrop and forge (zone) access permits.

Illustration of LIPS Gateway Mode. Now let go through the LIPS gateway mode with a complete example shown in Fig.6. During the initiation of host H_1, host H_1 authenticates itself to PS_1 and SG_1. It also issues a host access permit [2], $P^{host}_{Z_1 \to H_1}$, which it uses to authenticate packets from PS_1 and SG_1 back H_1, where $P^{host}_{Z_1 \to H_1} := H(IP_{PS_1}, K_t^{H_1})$, IP_{PS_1} is the IP address of PS_1, and $K_t^{H_1}$ is a secret key of H_1, which will be periodically refreshed. Similarly, host H_2 also authenticates itself to PS_2 and SG_2, and issues them a host access permit, $P^{host}_{Z_2 \to H_2}$.

In order to access host H_2 in zone Z_2, host H_1 (with an IP address of IP_1) sends a permit request to PS_1 to obtain a permit of H_2. On behalf of H_1, PS_1 contacts PS_2 (whose zone ID is IP_{PS_2}) if PS_1 does not have a permit for zone Z_2 yet. PS_1 finds PS_2 via a simple DNS convention, and authenticates with it by exchanging appropriate credentials. In response to the permit request, PS_2 returns a zone access permit, $P^{zone}_{Z_1 \to Z_2}$, to PS_1, where $P^{zone}_{Z_1 \to Z_2} = H(IP_{PS_1}, K_t^{Z_2})$, and $K_t^{Z_2}$ is a secret key of PS_2 (and shared with SG_2) at current time t. PS_1 mutates the received zone permit $P^{zone}_{Z_1 \to Z_2}$ into a host access permit, $P^{host}_{H_1 \to H_2} := P^{zone}_{Z_1 \to Z_2} \oplus H(IP_1, IP_2, K_t^{Z_1})$, where $K_t^{Z_1}$ is a secret key of PS_1 (and shared with SG_1) at current time t, and returns it to host H_1. PS_1 also *caches* the zone access permit $P^{zone}_{Z_1 \to Z_2}$. Upon receipt of the host access permit, H_1 caches the permit in its permit cache as an *timed* entry $[IP_2, P^{host}_{H_1 \to H_2}]$.

When host H_1 sends a packet to host H_2, it looks up its permit cache, attaches $P^{host}_{H_1 \to H_2}$ as a destination access permit in the LIPS packet, and $P^{host}_{Z_1 \to H_1}$ as the source access permit. When this packet reaches the security gateway SG_1, it verifies the destination access permit based on the source and destination IP addresses, IP_1 and IP_2, and its own secret key, $K_t^{Z_1}$. If the authentication succeeds, it restores the destination access permit to the destination *zone* access permit by performing $P^{host}_{H_1 \to H_2} \oplus H(IP_1, IP_2, K_t^{Z_1})$. Then SG_1 updates its LIPS

[2] A separate access permit can be issued to PS_1 and SG_1, using, e.g., their respective IP addresses. For simplicity of exposition, we treat PS_1 and SG_1 as if they were a single unit, responsible for the secure control and data plane operations, respectively.

cache by *refreshing* the entry $[IP_1, P^{host}_{Z_1 \to H_1}]$, and *replaces* the source access permit in the packet to the source *zone* access permit, $P^{zone}_{Z_2 \to Z_1} := H(IP_{PS_2}, K_t^{Z_1})$.

When the packet reaches the destination security gateway SG_2, it first verifies the destination access permit. If the authentication succeeds, SG_2 looks up its permit cache using the destination IP address, IP_2, and replaces the destination access permit in the LIPS packet with host H_2's access permit $P^{host}_{Z_2 \to H_2}$. In the meantime, it also mutates the source access permit in the LIPS packet into the source access permit, $P^{host}_{H_2 \to H_1} := P^{zone}_{Z_2 \to Z_1} \oplus H(IP_2, IP_1, K_t^{Z_2})$. (This last step can be optional if SG_2 and H_2 are located in a secure network for performance concerns.) Finally, when the LIPS packet reaches H_2, it verifies the destination access permit. If the authentication succeeds, it caches the source access permit in its permit cache with a (timed) entry $[IP_1, P^{host}_{H_2 \to H_1}]$. When H_2 wants to send a packet back to H_1, it follows the same procedure described above, reversing the role of host H_1 and host H_2.

3.3 Advantages and Limitations of LIPS Design

The design and implementation of LIPS have several advantages. As noted earlier, a key feature of LIPS is that *no secret* is shared across network domains, which makes the architecture more scalable and flexible[3]. Packet authentication is performed using only information carried in the header of a packet and *secret keys* held *locally* by security gateways and hosts. (Note that permit servers and security gateway do need share secret keys. However, they are conceptually two facets, *control* vs. *data* plane of the same security unit, and often can be implemented in a single box.) Thus packet operations can be done efficiently – our initial experimental testing in Section 4 shows that even with *software* implementation on common Linux platforms, it can be done at *near line speed*. Apart from maintaining their own secret keys, the only other information hosts need to maintain is an access permit cache: a security gateway or host maintains access permits for destination zones or hosts, respectively, with which it is currently communicating. The size of such caches are much smaller than routing tables in today's routers. Our architecture is also *incrementally* deployable. First, it is purely *edge-to-edge* (or "end-domain-to-end-domain"), as it does not require intermediate networks for assistance. Furthermore, only those hosts that need to be secured have to be patched with simple protocol enhancement, i.e., the *HAL* layer and LIPS permit servers, and to be placed "behind" security gateways for authentication and protection. In addition, *no*

[3] In his keynote at the ACM SIGCOMM 2003 Conference, Prof. David Cheriton of Stanford University succinctly summarizes the challenges in designing secure large-scale distributed systems: "trust $\neq$ security $\neq$ encryption" and "secret does not scale".

modification to applications is required.

Limitations. Since our goal is to get rid of the huge volume of most common unwanted packets on the current Internet, as a fist-line of defense, we choose a weak-form packet-origin authentication in LIPS. As a result, when attackers have the *accesses to the forwarding paths* of LIPS packets, they may conduct active attacks, such as payload replacing and replay permits. Since applications always use their own security schemes (e.g., SSL) for confidentiality and integrity, these active reply attacks are taken care of by these schemes, and their damages are limited to mostly the waste of bandwidth.

We have developed several schemes in LIPS to increase the difficulties of active attacks and minimize their damages. First, through separate *zone-level* and *host-level* access permits, LIPS isolates "bad" packets originating in one's own zone from those outside, and limit the abilities of attacks to mostly "man-in-the-middle" active *replay* attacks by real-time "sniffing" permits. Given that most attacks today are launched by end users, such attacks can be isolated within their originating domains, and can be more easily tracked down and taken care of. In addition, the "man-in-the-middle" replay attacks (such as replacing payload and replaying packets) in intermediate domains are in general much difficult to launch, as border gateway routers are typically connected via high-speed fiber optical links, and they are extremely hard to gain access to. Furthermore, to replay a valid permit, an attacker must have *accesses to the forwarding paths* of LIPS packets since a permit is domain- or host-specific; he also has to *real-time* sniff the permit since it is valid for a short period. We will examine the limited damages of these attacks in Section 4. Moreover, permit- or packet-replay attacks can be further mitigated by including, e.g., sequence numbers or random bits, in the security parameter fileds of access permits. For example, when a replay attack between zones is detected, LIPS will enable a sequence number in a zone permit such that attack traffic can be easily identified. As such a sequence number is for a zone instead of a flow, a security gateway only needs to maintain a limited amount of states. Lastly, by augmenting LIPS with active monitoring and rapid response defense mechanisms, we can quickly detect and throttle such attacks (e.g., by detecting duplicate access permits and adjusting timed keys). With such mechanisms, replay attacks will have only *localized* effect for a short period of time, with only "sniffed" hosts/domains being affected, due to the host-specific/domain-specific feature of access permits. We are further investigating active defense schemes that utilize the preliminary traffic accountability provided by LIPS.

Permit request floods may be another potential threat. However, as discussed in [31], attackers always have chances to attack the first handshaking of trust management procedure in an open distributed system. To our best knowledge, no effective methods can totally prevent such attacks. We use a few mechanisms to deal with permit request attacks. First, a security policy at a host

or a permit server limits who can issue permit requests to it. Therefore, only legitimate hosts/domains can issue permit requests. For host permit requests from a local domain, a host can monitor the request rates and identify attacks from its local domain. Since we have the complete control of the local domain, we can easily trace back and disconnect attack hosts. For domain permit requests, we require that each permit server uses a pair of public/private keys for its first zone authentication. In the case that a permit server is compromised, we can detect its request flood attack by monitoring its request rate, and deny its requests by setting up a filter or set a request rate limit until it is back to normal. In summary, all current open distributed systems suffer the problem in the trust initialization phase of control plane. Various intelligent methods have been proposed to mitigate this issue, but no methods can completely solve the problem as long as we allow unknown sources to initialize the handshaking procedure.

4 Performance Evaluation

In this section, we first evaluate the basic overhead of the LIPS framework itself, and then examine the effectiveness of LIPS in protecting server resources.

4.1 Basic LIPS Performance: Operation Overhead

Permit Generation and Verification Delay. The main cost in permit generation and verification is to compute a secure hash value using HMAC-MD5 [25]. It is performed twice for each LIPS packet in the sender's HAL for generating a source access permit and in the receiver's HAL for verifying a destination permit. The mean delay of generating a secure hash value in our implementation is about 3190 clock cycles, i.e., 1.14μs on a 2.8GHz Pentium running Linux. Using this measurement, we can estimate the mean response delay of permit requests. Assume that permit requests arrive at a host as a Poisson process with a mean arrival rate λ. Using the M/D/1 queueing model [32], the average permit processing time (including queueing delay) is $\frac{2-\lambda/\mu}{2\mu(1-\lambda/\mu)}$. Plugging in the above measurement as the permit generation rate μ, and assume the offered load λ/μ is as high as 0.95, we have a average permit processing delay of 10.5 μs, i.e., we can process 95K permit requests per second on a common PC. With such a high request processing rate, combined with a rate limit scheme and multiple high-end PCs to support the operations, we can easily handle a high volume of requests to mitigate request flooding attacks.

It also tells us that we can authenticate 95K packets per second at a destination (domain) with a Poisson input. With the average packet size of 844 bytes [33],

Table 2. HMAC Computation and Related.

Computing HMAC	Permit Request Process	Authentication Bandwidth	Permit Mutation Demutation
1.14 μs	95K per sec	640 MBps	815K per sec

Table 3. Memory Cost and Lookup Delay.

	Uniform	Pareto	UMN	WorldCup
M_{cache} (Mbytes)	3.1	4.6	1.6	2.2
D_{lookup} (cycles)	270	280	158	177

Table 4. Comparison: with and without LIPS over a dedicated link.

	Effective Bandwidth	Loss Rate	Jitter
With LIPS	90.7 Mbps	0.005%	0.025ms
W/O LIPS	93.7 Mbps	0.005%	0.022ms

our LIPS implementation can authenticate traffic at a rate around 640 Mbps on a common Linux PC, far beyond a common user's requirement.

Computing HMAC is also the main cost in permit *mutation/demutation.* The measured mean delay of mutation/demutation is about 3433 clock cycles in our implementation, i.e., a rate of 815,613 packets per second. Table 2 summarizes all computation related to HMAC. It also shows that *we can achieve much better performance when using a high-end PC.*

Insertion/Lookup Delay and Memory Cost of Permit Cache. As introduced in Section 3.1, we use a linear hashing scheme with controlled splitting to manage a permit cache. Let denote the mean delay of permit lookups/insertions delay as D_{lookup} and the memory cost of a permit cache as M_{cache}. We use two traffic models (uniform and pareto) and two real traces (UMN and WorldCup) as the input of our simulations to examine our design. Trace UMN is a real packet trace from a subnet at the University of Minnesota, which includes 90 hosts across a period of 40 days. Trace WorldCup is a web server trace from World Cup 98 site in its busiest day [34], including 73 millions web requests and more than two millions different destinations. We refer interested readers to [29] for the details of these models and traces. As summarized in Table 3, our permit cache management schemes performs reasonably well under both theoretical models and real traces.

Overall Overhead of LIPS. We conduct experiments to measure the overall overhead introduced by our LIPS implementation in data transmission, compared with IP. In these experiments, we use Iperf [35] to send an CBR UDP flow from a host to another via a dedicated 100 Mbps link. When the CBR rate is lower than 100 Mbps, there are almost no differences between the transmissions with or without LIPS. Table 4 shows the Iperf measurements when the CBR rate is 100 Mbps. Even in this stress test, the difference between the

transmission bandwidth with LIPS and that without LIPS is negligibly small, about 3%.

4.2 Effective LIPS Protection

We use simple analytical models to show how LIPS helps stop DoS attacks from two aspects: the chances for zombies to start DoS floods and the probabilities of successful attacks. We focus on the replay of host permits in LIPS domains because it is rather difficult to gain access to inter-domain links to sniff a domain permit [4] . The real time and host-specific nature of permits dramatically increases the difficulty to generate attacking traffic. Furthermore, a fast response mechanism helps us quickly stop floods. Therefore, it is extremely difficult to bring down a LIPS-protected target via permit replay.

LIPS significantly reduces the chances for zombies to spoof IP addresses and generate permit-replay floods. We first examine the spoofing chances for a zombie in a LIPS domain. Assume that an egress filter is deployed for the domain. Hence a zombie can only spoof IP addresses in the domain. Under IP with ingress/egress filtering[5], a zombie can spoof any IP address in the domain with a probability of 1, once it finds out which addresses are allowed to access a target. Under LIPS, the chances of IP spoofing is significantly limited. Since each host permit has an effective period and is domain-specific, to sniff host permits for spoofing, a zombie must have access to the path to a destination in real time. In addition, because each permit is only valid for a short period of time in a specific domain, a zombie has to real time sniff a valid host permit to spoof/replay in the specific domain, and it is impossible for zombies to accumulate a large number of host permits ahead of time to launch flooding attacks. Permit replay is automatically stopped when there is no legitimate traffic to a destination.

We define p_z as the probability that a host is compromised as a zombie in a domain, and p_s as the probability that a zombie can sniff a valid permit to a target in the domain. Assume that a legitimate host communicates with a target server as a Poisson process, i.e., both the intervals between sessions and the durations of these sessions are exponentially distributed. As shown in Fig.7(a), given different p_z and p_s, the spoofing probabilities for zombies under LIPS are far lower than 1, the chance under IP with ingress/egress filtering. We choose the mean arrival rate as two sessions per minute and the mean duration as three seconds in these tests.

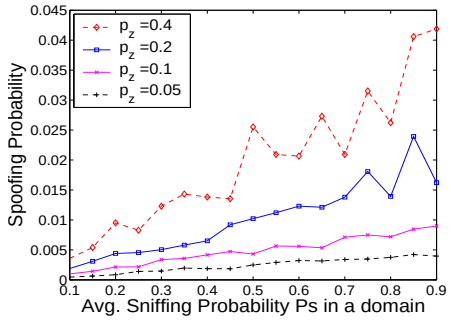
[4] To defeat domain-permit-replay attacks, we use a security association between two domains plus a sequence number to deal with the replay of domain permits. Since this type of attack is very unlikely to occur, we only activate the protection scheme when a domain permit replay attack is detected.

Consequently, LIPS dramatically suppresses zombies' capabilities to launch flooding attacks to a target. To avoid being easily detected and taken out, assume each zombie only spoofs an IP address by replaying a sniffed valid permit at the similar rate of a legitimate flow. Under LIPS, a zombie will not have a valid permit to replay when it can not find an active permit. While under IP with ingress/egress filtering, a zombie can spoof a source at any time. Assume we have a domain of 100 hosts; those hosts communicate with a remote server as Poisson processes; the mean flow rate of a legitimate session is 128Kbps. Fig.7(b) shows that the aggregate flooding bandwidth to the server, which can be generated by zombies in the domain. The top four lines are flooding rates under IP with ingress/egress filtering at various p_z, while the bottom four lines are flooding rates under LIPS with the same conditions. Note that the Y-axis is in a log scale. Clearly, it is very difficult for zombies to generate sufficient traffic to flood the server under LIPS; while it is fairly easy under IP with ingress/egress filtering. Furthermore, Fig.7(c) shows the ratio of the aggregate flooding capability of the above domain under IP with ingress/egress filtering to that under LIPS. Apparently, LIPS significantly reduces the flooding capability, especially when p_s is small. In addition, Fig.7(d) shows that an attacker needs about 10^4 to 10^5 domains as the above to flood a LIPS-protected 1 Gbps link with over 100% unwanted packets. It is extremely difficult for an attacker to collect such huge amount of resources. Besides, when we have multiple incoming links for a protected server, it will be even more difficult for such attacks to be successful.

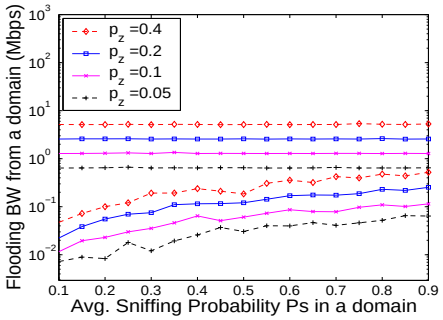
Furthermore, we use a simple Stochastic Knapsack framework to model a DoS attack to a protect incoming link of a target [19]. Assume a DoS attack is launched by a set of zombies at various times, and each zombie launches its attack independently, e.g., attack codes is activated by user operations, say opening a file or email. We use C to denote the total amount of incoming bandwidth available. Assume legitimate flows (or attacking flows) have an exponential arrival rate with a mean of λ_l (or λ_a), a bandwidth requirement b_l (or b_a), and an exponential service time with a mean of μ_l (or μ_a). The system admits an arrival whenever bandwidth available. In this model, the probability of a successful DoS attack is the blocking probability corresponding to the legitimate traffic, defined as follows:

$$P_b = 1 - \frac{\sum_S (\rho_l^{n_l}/n_l!) \cdot (\rho_a^{n_a}/n_a!)}{\sum_{S'} (\rho_l^{n_l}/n_l!) \cdot (\rho_a^{n_a}/n_a!)}$$

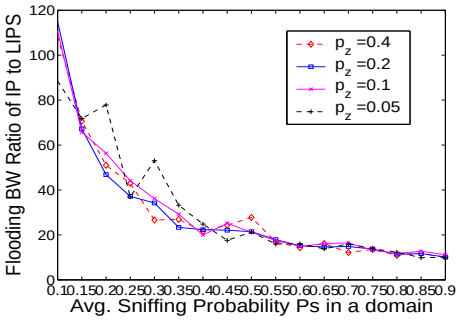
where S is the set of cases that an arriving legitimate flow can be admitted, and S' is the set of cases that either a legitimate flow or an attacking flow is admitted; in each case, n_l is the number of legitimate flows admitted, and n_a is the number of attacking flows admitted; and offered load $\rho_l = \lambda_l/\mu_l$, $\rho_a = \lambda_a/\mu_a$. Here we assume $b_l = b_a$, since zombies are in the same population as



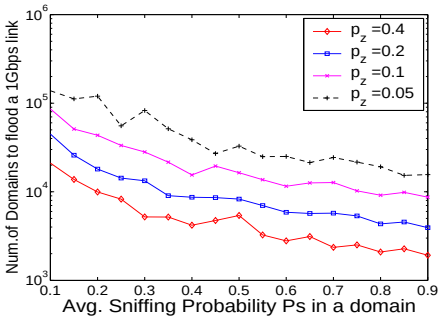
(a) Spoofing Probabilities in LIPS for a zombie.



(b) Comparison of Aggregate Flooding Bandwidth of a domain: IP with ingress/egress filtering vs. LIPS.



(c) Flooding Bandwidth Ratio of IP with ingress/egress filtering to LIPS.



(d) Number of domains needed to flood a 1 Gbps destination link.

Fig. 7. LIPS significantly reduces spoofing chances and restricts flooding capability.

the legitimate users [19]. The load level of attack traffic has to be significantly higher than that of legitimate traffic in order to blocking legitimate traffic. Fig.8 shows the blocking probability of legitimate flows as we increase the load of attacking traffic. We choose $C = 100$ Mbps, $b_l = b_a = 1$ Mbps, and legitimate load $\rho_l = 1$. To block 90% of legitimate traffic, the attacking load has to be 1000 times heavier than the legitimate traffic.

Stopping Permit-Replay Flooding Attacks in LIPS domains. Since zombies have to generate very high attacking load to launch successful attacks as shown in the above, it is easy to identify them at source domains through traffic monitoring schemes, and then isolate them through local defense mechanisms, e.g., instantly reconfiguring filters at routers to drop all packets from these zombies and taking further actions later with more advanced approaches. We can also detect flooding attacks at a destination domain and inform a source domain to fix corresponding zombies through inter-domain collaboration. In this case, we first detect a flood at a destination domain via, e.g., observing a sharp increase of packets from a source or using a bloom filter to detect replayed packets. Then, the destination permit server (PS) informs the source PS this event through their security association. Once the source PS confirms this event, it revokes the host permit at its SG(s) that automatically

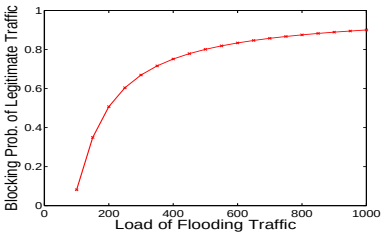


Fig. 8. Blocking Probability of legitimate flows as the attacking traffic load increases.

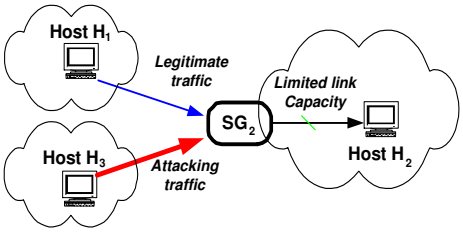
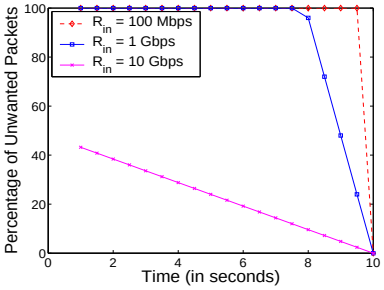
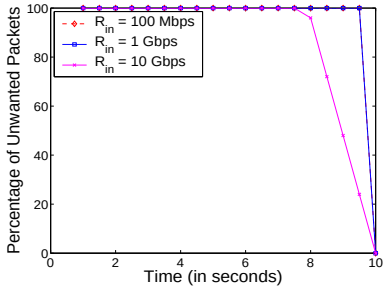


Fig. 9. Preventing flood from non-LIPS domains.



(a) 1,000 replaying sources: 100 sources in each of 10 domains.



(b) 10,000 relaying sources: 100 sources in each of 100 domains.

Fig. 10. Static Attacks: Delays to shut off all replaying sources.

drops outgoing packets with the specific host permit; it also can instantly reconfigure filters at local routers to drop packets from the zombie.

We use the following example to show the effectiveness of the above mechanism in protecting a server and its critical link at a destination domain, which services clients from N_d domains. We assume that the incoming link of the server has a rate of R_{in}, say, 100 Mbps, 1Gbps, or 10 Gbps, respectively; the delay of shutting off a zombie is exponential distributed with a mean D_f. Note that different source domains can take actions parallelly. Assume a zombie has an outgoing link 10 Mbps and can generate at a rate of R_z, say, 400 packets per second. Assume that no new zombies are added after an attack is started. The load on the incoming link of a server is defined as $L_0 - L_r(t)$, where $L_0 = R_z \cdot N_z \cdot N_d$, where N_z is the average number of zombies in a source domain; $L_r(t)$ is the reduced load by revoking zombies at source domains and it is determined by N_d, R_z, t, and D_f; and t is the time after the attack is started. As shown in Fig. 10(a) and Fig. 10(b), we can quickly shut off these replaying source in 10 seconds, with $D_f = 0.1$ second.

Stopping flood attacks from non-LIPS domains. Here we use a simple experiment setting to show how the LIPS protects a incoming link of a server from flooding by traffic from non-LIPS domains. The experimental setting is shown in Fig.9. Host H_1 transmits a real-time CBR flow F_l to host H_2, e.g., a surveillance video stream, while host H_3 tries to flood H_2 with a CBR

Table 5. Measured Bandwidth (in Mbps).

Attacking Rate	Flow F_l Without LIPS			Flow F_l With LIPS		
	0.4Mbps	1.0Mbps	1.8Mbps	0.4Mbps	1.0Mbps	1.8Mbps
1.0 Mbps	0.400	1.000	1.290	0.400	1.000	1.800
2.0 Mbps	0.315	0.693	0.92	0.400	1.000	1.800
4.0 Mbps	0.208	0.376	0.667	0.400	1.000	1.800
6.0 Mbps	0.164	0.275	0.489	0.400	1.000	1.800
8.0 Mbps	0.149	0.199	0.371	0.400	1.000	1.800

Table 6. Measured Packet Loss (Percentage).

Attacking Rate	Flow F_l Without LIPS			Flow F_l With LIPS		
	0.4Mbps	1.0Mbps	1.8Mbps	0.4Mbps	1.0Mbps	1.8Mbps
1.0 Mbps	0	0	27	0	0	0
2.0 Mbps	21	30	49	0	0	0
4.0 Mbps	48	62	63	0	0	0
6.0 Mbps	59	73	73	0	0	0
8.0 Mbps	63	80	79	0	0	0

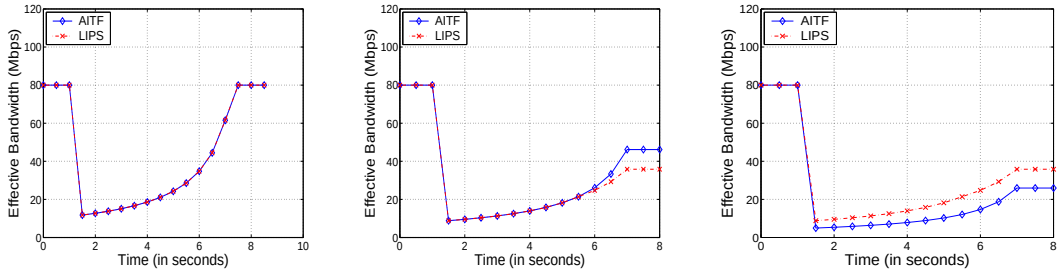
Table 7. Packet Jitter (in milliseconds).

Attacking Rate	Flow F_l Without LIPS			Flow F_l With LIPS		
	0.4Mbps	1.0Mbps	1.8Mbps	0.4Mbps	1.0Mbps	1.8Mbps
1.0 Mbps	0.011	0.005	8.79	0.016	0.010	0.024
2.0 Mbps	9.08	19.78	20.3	0.006	0.007	0.049
4.0 Mbps	1.60	19.33	12.93	0.024	0.034	0.023
6.0 Mbps	1.98	4.67	12.54	0.017	0.008	0.100
8.0 Mbps	8.64	18.7	4.96	0.008	0.011	0.027

non-LIPS attacking traffic. We set the capacity of H_2's incoming link to 2 Mbps using Linux CBQ. We tested three different rates of F_l at 0.4, 1.0, and 1.8 Mbps under five attacking rates at 1.0, 2.0, 4.0, 6.0, and 8.0 Mbps. When the total traffic rate is higher than the link capacity capacity, without LIPS, we see significant damages on F_l in bandwidth, packet losses, and packet jitters, as shown in the second super-columns of Table 5, Table 6, and Table 7, respectively. On the contrary, when we use a LIPS gateway protect the link, flow F_l was able to reach H_2 at its required bandwidth with no packet losses and negligible jitters, as shown in the third super-columns in the tables.

Comparing LISP with AITF. Both AITF and LIPS are designed as simple enhancements of current systems to distributedly filter out common flood packets. They both are subject to advanced active replay attacks if attackers can access their packet forwarding paths, and are dependent on in-depth security schemes to address this issue and achieve other security requirements.

There are a few differences between LIPS and AITF. First, LIPS is an edge-



(a) All domains cooperate.

(b) 75% of end domains cooperate, and all intermediate domains cooperate.

(c) 75% of all domains cooperate.

Fig. 11. Comparing the effective bandwidth of LISP with that of AITF: given the same number of attack sources.

to-edge approach, while AITF is an infrastructure-like cross-domain approach that requires the changes of a significant portion of intermediate routers. Exploiting the trust of these intermediate routers in the forwarding paths, AITF provides more filtering power on forwarding paths across intermediate domains. However, this protection comes with a price as discussed in [23], due to its filtering escalation. The filtering escalation may cause collateral damages when both legitimate traffic and attack traffic share the same intermediate routers that do not cooperate, i.e., AITF may block all packets from non-cooperative routers. In addition, the overhead of route records in AITF is about 10% of payload and grows with the increase lengths of forwarding paths, e.g., if a packet goes through 8 hops, a packet carries a route record of 87 bytes. A LIPS header is fixed as 24 bytes.

We conduct simple tests to compare the effective bandwidth of LIPS and AITF. The results show that, when all intermediate routers cooperate, AITF performs better than LISP; when intermediate routers do not always cooperate, LISP performs better. Assume that 60,000 attack sources aim at a victim with a total of 600 Mbps attack traffic. The victim's gateway has an OC-48 ingress link of 2.488 Gbps, and the victim has an ingress link of 100 Mbps from the gateway. Legitimate sources are co-located with attack sources in source domains. When no attacks, the victim has an effective bandwidth of 80 Mbps. Once detecting attacks, the victim can send 10,000 filter requests per second to its gateway; its gateway is able to filter attack flows and request the gateways at sources domains to stop attack flows. Each gateway can filter out 10,000 flows for the victim. In the first case, we assume that all gateways cooperate with the victim gateway to block attack flows. Fig.11(a) shows that the effective bandwidth of the victim under LIPS is generally the same as that of AITF. In the second case, when 75% of end-domain gateways cooperate and *all* intermediate gateways cooperate, as shown in Fig.11(b), AITF performs better than LIPS because the intermediate routers provide more filtering power without collateral damages. In the third case, when 75%

of all source and intermediate gateways cooperates, as shown in Fig.11(c), LIPS performs better than AITF, because of the collateral damages caused by some intermediate routers in AITF.

5 Conclusions

LIPS is a simple packet authentication mechanism which provides preliminary traffic accountability for restricting most common unwanted traffic. We presented the basic design of LIPS and a prototype implementation on a Linux platform. Our analytical, simulation and experimental results show that LIPS is capable of confining common spoofed and unsolicited packets with light overheads. It can be incrementally deployed on a large scale with small patches to end hosts and no changes in intermediate routers. Currently, we are incorporating active monitoring and rapid-response defense mechanisms into LIPS for exploiting its preliminary traffic accountability to further improve its effectiveness. In the meantime, we are also developing an overlay inter-domain service to protect the traffic between two end domains. Combining the inter-domain protection with LIPS, we have a more complete scheme to deal with common spoofing and flood traffic.

References

[1] D. Clark, The design philosophy of the DARPA Internet protocols, in: Proc. ACM SIGCOMM, 1988.

[2] S. Kent, R. Atkinson, Security architecture for the internet protocol, RFC2401, Nov. 1998.

[3] R. Ramanujan, R. Ramanujan, M. Kaddoura, J. Wu, C. Sanders, K. Millikin, Vpnshield: Protecing vpn services from denial-of-service (dos) attacks, in: DARPA Information Survivability Conference and Exposition, 2003.

[4] D. Harkins, D. Carrel, The internet key exchange (IKE), RFC2409, Nov. 1998.

[5] P. Ferguson, D. Senie, Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing, RFC 2267, Jan. 1998.

[6] A. Bremler-Barr, H. Levy, Spoofing prevention method, in: Proc. IEEE INFOCOM'05, Mar. 2005.

[7] G. Hadjichristo, N. D. IV, C. Midki, Ipsec overhead in wireline and wireless networks for web and email applications, in: Proc. IEEE IPCCC, Apr. 2003.

[8] S. Miltchev, S. Ioannidis, A. D. Keromytis, A study of the relative costs of network security protocols, in: Proc. USENIX Annual Conf. Freenix Track, Jun. 2002.

[9] D. Estrin, J. C. Mogul, G. Tsudik, Visa protocols for controlling inter-organization datagram flow, IEEE Journal on Selected Areas in Comm., May, 1989.

[10] IETF, Better than nothing security, in: http://www3.ietf.org/proceedings/05mar/btns.html, Mar. 2005.

[11] R. Moskowitz, P. Nikander, P. Jokela, T. Henderson, Host identity protocol, http://www.ietf.org/internet-drafts/draft-ietf-hip-base-01.txt, Oct. 2004.

[12] G. Montenegro, C. Castelluccia, Statistically unique and cryptographically verifiable (sucv) identifiers and addresses, in: Proc. ISOC Symposium on Network and Distributed System Security, 2002.

[13] J. Ioannidis, S. Bellovin, Implementing pushback: Router-based defense against ddos attacks, Proc. Network and Distributed Systen Security Symposium, 2002.

[14] T. Anderson, T. Roscoe, D. Wetherall, Preventing internet denial-of-service with capabilities, in: Proc. Hotnets'03, Nov. 2003.

[15] M. G. Gouda, E. N. Elnozahy, C.-T. Huang, T. McGuire, Hop integrity in computer networks, IEEE/ACM Transactions on Networking, Jun. 2002.

[16] H. Wang, A.Bose, M.Gendy, K.Shin, IP Easy-pass: Edge Resource Access Control, in: IEEE INFOCOM'04, 2004.

[17] J. Li, J. Mirkovic, M. Wang, P. Reiher, L. Zhang, Save: Source address validity enforcement, in: Proc. IEEE INFOCOM, 2002.

[18] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Practical network support for IP traceback, Proc. of ACM SIGCOMM, pp. 295-306, Stockholm, Sweden.

[19] A. Keromytis, V. Misra, D. Rubenstein, SOS: Secure overlay services, Proc. ACM SIGCOMM'02.

[20] D. Aderson, Mayday: Distributed filtering for internet services, in: 4th Usenix Symposium on Internet Technologies and Systems, Seattle, Washington, Mar. 2003.

[21] D. Adkins, K. Lakshminarayanan, A. Perrig, I. Stoica, Towards a more functional and secure network infrastructure, UCB Technical Report No. UCB/CSD-03-1242.

[22] A. Juels, J. Brainard, Client puzzles: A cryptographic countermeasure against connection depletion attacks, in: Proc. of NDSS '99 (Networks and Distributed Security Systems), pages 151-165, 1999.

[23] K. Argyraki, D. R. Cheriton, Active internet traffic filtering: Real-time response to denial-of-service attacks, in: USENIX Annual Technical Conference, April 2005.

[24] A. Menezes, P. Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, ISBN: 0-8493-8523-7, 1996.

[25] M. Oehler, R. Glenn, Hmac-md5 ip authentication with replay prevention, RFC2085, Feb. 1997.

[26] D. J. Bernstein, Syn cookies, http://cr.yp.to/syncookies.html.

[27] M. Blaze, J. Feigenbaum, J. Ioannidis, A. Keromytis, The keynote trust management system, Version 2. RFC-2704.

[28] W. Litwin, Linear hashing: A new tool for file and table addressing, in: Proc. VLDB, 1980.

[29] C. Choi, Y. Dong, Z.-L. Zhang, Design and prototype of lightweight internet permit system, Technical Report, Dept. of Computer Science, Univ. of Minnesota, 2004.

[30] B. Neuman, T. Tso, Kerberos: An authentication service for computer network, IEEE Comminucation Magazine,Sept 1995.

[31] K. Argyraki, D. Cheriton, Network capabilities: The good, the bad and the ugly, in: Proc. of ACM Hot Topics in Networks (HotNets) Workshop, College Park, MD, Nov. 2005.

[32] R. K. Jain, The Art of Computer Systems Performance Analysis, John Wiley and Sons, Apr. 1991.

[33] Sprint ipmon dms-packet size distribution, http://ipmon.sprint.com/packstat/, Apr. 2003.

[34] 1998 world cup web site access logs, http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[35] A. Tirumala, et. al., Iperf: Tools for measuring tcp/udp performance, http://dast.nlanr.net/Projects/Iperf, 2003.