Enhancing Security and Resiliency over Wired and Wireless Networks

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Changho Choi

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. Zhi-Li Zhang, Adviser

August 2007

To the Lord My God and My Family

Acknowledgments

Give Thanks to The Lord My God!

Ebenezer, "Thus far has the Lord helped us." (1 Sam 7:12)

It is a great pleasure to thank many people who have helped me to successfully finish my degree. First of all, I would like to thank my advisor, Professor Zhi-Li Zhang, for his support and guidance during my study at University of Minnesota. His passion and vision always encourage and motivate me to critically think and extend researches. I also thank to my preliminary and final oral examination committee members, professor Yongdae Kim, professor Andrew Odlyzko, professor Abhishek Chandra, and professor Vipin Kumar, for giving me their valuable suggestion. Their efforts have surely made this thesis a lot better.

I want to thank Dr. Yingfei Dong, Mukund Srinivasan, and Guor-Huar Lu who had done several projects together. I would not make this thesis mature without discussion with them. I thank to my dear fellows in multimedia networking research group: Jaideep Chandrashekar, James Beyer, Yingping Lu, Kuai Xu, Yinzhe Yu, Sanghwan Lee, Ewa Kusmierek, Baek-Young Choi, Sejun Song, Zhenhai Duan, Srihari Nelakuditi, Aravindan Raghuveer, Esam Sharafuddin, Hunjung Kang, Yu Jin, Hal Peterson, Sourabh Jain, Ting Zhu, Sarah Sharafkandi, and Sungil (Joseph) Hong. I am grateful for their generous help and interests.

Also I would like to thank Dr. Allalaghatta Pavan, Dr. Haowei Bai, Dr. Srivatsan Varadarajan, and Rakesh Jha, for providing me with a valuable internship opportunity at Honeywell Laboratory. With their help, I could have a precious experience in working on practical projects, which help me to expand my research interests.

I am very grateful that I have a great congregation who prays for me and encourages me whenever I am down. I give special thanks to following people but not limited to them: pastor Abraham Yoonsik Park, pastor Mansul Kim, pastor Sungeun Kim, pastor Jiyong Park, deacon Choonsik Jhun, deacon Giljae Lee, deacon Hyungho Han, deacon Moonsu Han, Sunghwa Kwon, Jaehoon Jeong, Sangyeob Sung, Eunhye Yu, Junghan Bae, Yoonhee Sung, Jungju Hong, and Minzee Kim.

It has been a great pleasure to meet many Korean colleagues at University of Minnesota and I want to thank them including Seonho Kim, Manki Min, Jinsung Yoo, Sangho Kim, Jinpyo Kim, Sungjae Kim, Jinoh Kim, Myunghwan Park, and Taehyun Hwang. From time to time we discussed research issues and our concerns. Also we played tennis together.

Although many people who have generously helped me during my study may not be listed here, I would like to thank everyone of them and wish the best for them.

I am very lucky to have a great family: my parent, brothers, sisters, mother-in-law, brothers-in-law, and sisters-in-law. With their love, prayer and support, I could successfully finish my study here.

I appreciate my dearest wife, Unjung Kim, with all my heart. She always trusts and encourages me in everything. Without her endless love and self-sacrificing, I would never make this done. I also thank my little angel, Eunjee. She has always given delight to us.

Abstract

As computer networks grow in size and range, demand for enhanced security and resiliency increases and becomes main concern over wired and wireless networks. In this dissertation we address security and resiliency issues in control plane: security over wired networks and resiliency over wireless networks.

The success of the current Internet is mainly attributed to openness and distributed control. However this open and distributed nature of the Internet makes the protection of the entire Internet resources nearly impossible. In addition the cost of such a solution would 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 thesis we focus on enhancing security in terms of securing and protecting critical Internet resources from unauthorized accesses. We propose two systems, secure name service (SNS) and lightweight Internet permit system (LIPS), that make the current Internet more secure from unauthorized accesses and DoS attacks. SNS provides a scalable and flexible framework for establishing trust (and thereby also accountability) among networks and hosts. SNS secures and protects the critical resources through *resource virtualization* and *authenticated packet forwarding*. LIPS ensures traffic accountability through *lightweight access permits* for stopping unwanted packets. Only packets with valid access permits may go through security gateways and other packets are automatically filtered out. Our prototype implementation and experiments have demonstrated the feasibility of deploying SNS and LIPS on Linux systems.

We also develop a scalable, opportunistic wireless routing protocol called *bubble routing protocol* (BRP) to enhance resiliency over wireless networks. Unlike wired networks, links in wireless networks are failure prone, unstable, and probabilistic. This probabilistic link characteristic in wireless networks makes the usage of preselected path inappropriate for reliable communication without sacrificing performance. Furthermore wireless networks broadcast in nature and this broadcast nature can exploit rich (path) spatial diversity not used in most existing routing protocols in wireless networks. BRP utilizes neighbor nodes and gives higher chances to better opportune receivers for forwarding packets towards destinations. BRP implicitly uses multiple paths that localize the impact of local instability and failure.

Contents

List of Figures xi

List of Tables xii

1 Introduction 1

1.1 Motivation **1**

1.1.1 Enhancing Security over Wired Networks 1

1.1.2 Enhancing Resiliency over Wireless Networks 3

1.2 Contributions of This Dissertation **5**

1.2.1 Secure Name Service (SNS) 5

1.2.2 Lightweight Internet Permit System (LIPS) 6

1.2.3 Bubble Routing Protocol (BRP) 7

1.3 Organization **9**

I Enhancing Security over Wired Networks 10

2 SNS: Secure Name Service 11

2.1 Introduction **11**

2.2 SNS Architecture and Components **15**

2.3 Secure Name Service (SNS) **18**

2.3.1 Secure Name Convention and SNS Identities 18

2.3.2 Components of SNS Naming System 19

2.3.3 SNS Name Resolution 23

2.4 Authenticated Packet Forwarding 26

2.4.1 Design of Authenticated Packet Forwarding 26

2.4.2 Prototype and Experimental Evaluation 31

2.5 Dynamic Table Management at SGs 37

2.5.1 Table Operations and Requirements 37

2.5.2 Dynamic Directory Schemes and Their Performance 39

2.6 Related Work and Discussion 48

2.7 Summary 51

3 LIPS: Lightweight Internet Permit System 52

3.1 Introduction 52

3.2 Basics of LIPS Architecture 55

3.2.1 LIPS Packet 55

3.2.2 Access Permit 56

3.2.3 Exchange of Access Permits between a Source and a Destination 58

3.2.4 Permit Cache 59

3.2.5 Key Management 60

3.3 LIPS Design and Implementation 61

3.3.1 LIPS Host Mode 61

3.3.2 Design of LIPS Gateway Mode 64

3.3.3 Advantages of LIPS Design and Implementation 71

3.4 Performance Evaluation 73

3.4.1 Basic LIPS Performance: Operation Overhead 73

3.4.2 Effective LIPS Protection 75

3.4.3 Stopping Flooding Attacks from non-LIPS Domains 81

3.4.4 Security Strength of Access Permits 83

3.5 Related Work 86

3.6 Summary 88

II Enhancing Resiliency over Wireless Networks 89

4 BRP: Bubble Routing Protocol without Graphs 90

4.1 Introduction 90

4.1.1 Related Work 92

4.2 Bubble Routing Protocol (BRP) 95

4.2.1 BRP Operation 95

4.2.2 Design Concerns 97

4.3 Route Metric Discovery 99

4.3.1 Dissemination of Route Metric Request (RMREQ) 100

4.3.2 Scheduling of RMREQ Messages 102

4.3.3 Handling Route Metric Reply (RMREP) 102

4.4 Data Delivery 106

4.4.1 States Maintained at Nodes 107

4.4.2 BRP Data Packet Header 107

4.4.3 Batch Preparation 108

4.4.4 Processing Data Packets 108

4.4.5 Scheduling 109

4.5 Greedy Set Selection (GSS) Algorithm 111

4.5.1 Square Area Communication Range 112

4.5.2 Greedy Set Selection (GSS) Algorithm 114

4.6 Comparative Study Based on Analytical Models 121

4.7 Performance Measurement 126

4.7.1 Regular Networks 127

4.7.2 Wireless Mesh Networks 130

4.8 Summary 133

5 Conclusion 134

Bibliography 137

List of Figures

2.1 SNS Framework 15
2.2 Process of Exchanging Security Parameters between two SNSs. 22
2.3 Resolving a Query via Local SNS 24
2.4 Resolving a Query across Domains 24
2.5 Two SNS-enabled Domains 27
2.6 Packet Formats at each Steps from Host *src* to Host *dst* 29
2.7 Format of Authenticator 32
2.8 Flow Chart of sIP Procedure 33
2.9 Experimental Setting 35
2.10 Multi-Level Directory 39
2.11 Single-Level Directory 39
2.12 Comparison of Delay and Memory Cost with Analytical Models 45
2.13 Comparison of Delay and Memory Cost using Simulations 46

3.1 Converting an IP Packet into a LIPS Packet. 56
3.2 Format of LIPS Packet Header. 56
3.3 Format of LIPS Permit. 57
3.4 LIPS Message Exchanges for Setting up Access Permit. 58
3.5 Illustration of LIPS Gateway Mode 65
3.6 Spoofing Probabilities in LIPS for a Zombie 78
3.7 Aggregate Flooding Bandwidth:IP with Ingress/Egress Filtering vs. LIPS 78

3.8 Potential Flooding Bandwidth Ratio of IP with Ingress/Egress Filtering to LIPS 79
3.9 Number of Domains Needed to Flood a 1 Gbps Destination Link 79
3.10 Blocking Probability of Legitimate Flows as Attacking Traffic Load increases 82
3.11 1,000 Replaying Sources: 100 Sources in each of 10 Domains 82
3.12 10,000 Replaying Sources: 100 Sources in each of 100 Domains 83
3.13 Preventing Flooding Attacks from non-LIPS Domains 84

4.1 Route Metric Discovery Message 99
4.2 BRP Data Packet Header 107
4.3 Communication Range of a Node 113
4.4 Network Area Coverage 114
4.5 Node Selection in GSS Algorithm 116
4.6 The Number of Bubbles Required with GSS 118
4.7 The Number of Rebroadcast Messages per Node 119
4.8 Ratio of the Number of Bubbles with GSS to S_o 119
4.9 Illustration of A_z and B_z 121
4.10 Probability of Routing/Forwarding Failure on Various Network Density . 124
4.11 Probability of Routing/Forwarding Failure on Various Link Failure Rate . 124
4.12 Number of Rebroadcast RMREQ Messages per Node on Regular Networks 129
4.13 Number of Suppressed RMREQ Messages on Regular Networks 129
4.14 Control Message Overhead on Regular Networks 130
4.15 Number of Rebroadcast RMREQ Messages per Node on Roofnet 131
4.16 End-to-End Packet Delivery Ratio on Roofnet 132
4.17 Control Message Overhead on Roofnet 132

List of Tables

2.1 Delays of MAC Generation 33
2.2 Delays of Forwarding Components 34

3.1 Permit Cache 60
3.2 Summary of Notations 66
3.3 HMAC Computation and Related 74
3.4 Memory Cost and Lookup Delay 75
3.5 Comparison: With and Without LIPS over a Dedicated Link 75
3.6 Measured Bandwidth 84
3.7 Measured Packet Loss 84
3.8 Packet Jitter 85
3.9 Lower Bound for Birthday Attack on Permits 85

4.1 Notations 100
4.2 Notations in GSS 115

Chapter 1

Introduction

1.1 Motivation

The proliferation of wired and wireless networks makes people connected to the Internet from any place at any time. The Internet works on control plane and data plane. Data plane is used for data delivery and strongly relies on control plane. Therefore control plane is much more critical in the current Internet. Control plane is mainly composed of security associated tasks and routing protocols. In wired networks security is a key issue while resiliency is a key problem in wireless networks. This thesis is composed of two crucial topics that enhance those key issues over wired and wireless networks. We develop two schemes that enhance security over wired networks in the first part. In the second part, we develop a routing protocol for enhancing resiliency over wireless networks.

1.1.1 Enhancing Security over Wired Networks

As we become more and more reliant on the Internet, the number of network security attacks with the aim to abuse or disrupt such services has also significantly increased. As reported by the Computer Emergency Response Team (CERT), the number of reported cyber attacks has doubled in recent years, aided partly by many hacker tools available on the Internet. [17] In addition the sophistication of cyber attacks has also increased. The emergence of massive distributed denial-of-service attacks is one such example. Because of the *decentralized* and *open* nature of the current Internet, it is extremely difficult, if not impossible, to protect the *entire* Internet from cyber attacks. In addition the cost of such a solution would be economically prohibitive due to the sheer size of the Internet. It is therefore important to selectively secure Internet services that are critical, namely those services that provide significant values. In enhancing the current Internet with security mechanisms, and evolving it into a more secure infrastructure, important trade-offs such as openness of the Internet, efficacy and flexibility of security protection, cost of deployment and ease of use must be carefully studied and evaluated.

In this thesis we develop two systems that make the current Internet more secure and resilient under cyber attacks. We focus on the challenges in enhancing security and resiliency of the Internet information services/resources. To ensure integrity of the Internet information services, *critical* resources such as web and other application servers, databases, data repositories and storage systems, networking and other resources must be protected from unauthorized accesses, intrusion, disruption, denial-of-service (DoS) attacks and other cyber threats.

We first devise a secure name service (SNS) that provides a scalable and flexible framework for establishing trust (and thereby also *accountability*) among networks and hosts, and for securing Internet community services *without* sacrificing their open and dynamic nature. We also propose *lightweight Internet permit system (LIPS)* that ensures traffic accountability through fast packet authentication for stopping unwanted packets. Both serves as a comprehensive "first-line of defense" against unauthorized accesses, intrusions, DoS attacks and other cyber threats by limiting the abilities of malicious users to launch attacks while hiding their identities. In addition to pro-active protection, we will also explicitly incorporate *active monitoring* and *rapid response* defense mechanisms into the proposed architecture for further securing critical Internet community services.

1.1.2 Enhancing Resiliency over Wireless Networks

The achievement of wireless technology proliferates the multi-hop wireless ad hoc networks all around the world. Mobile ad hoc network (MANET) is a well known multi-hop wireless ad hoc network with no fixed infrastructure. MANET can be fully deployed in applications such as disaster relief, tetherless conference rooms, and battlefield situations. Wireless mesh network is another emerging multi-hop wireless network with small mobility and is widely deployed in metropolitan areas. The proliferation of multi-hop wireless ad hoc networks increases the demand for a resilient and efficient routing protocol.

Many routing protocols have been proposed for multi-hop wireless networks. [60, 38, 61, 57, 41, 74, 42, 46, 36, 13] Most of them have been adopted from wired network routing protocols and use *pre-specified* single or multiple minimum cost paths. They use conventional graph theory to pre-specify paths. Links are represented by edges and a path by a sequence of edges.

However, unlike wired networks, links in wireless networks are failure prone, unstable, and probabilistic. Therefore conventional graph theory notion is inadequate for applying for wireless networks. These different link characteristics (e.g., probabilistic delivery and instability) in wireless networks also make a migration of existing routing protocols in wired networks to wireless networks difficult. In addition this probabilistic link characteristic in wireless networks restricts reliable communications when a pre-selected static path is used.Furthermore wireless networks have broadcast nature and this broadcast nature can exploit rich (path) spatial diversity that is not applied to most existing routing protocols in wireless networks.

In this dissertation we develop a scalable and opportunistic bubble routing protocol (BRP) that does not use the conventional graph theory. Furthermore BRP does not pre-select a single or multiple paths. Instead of designating a next hop node and path(s), BRP utilizes neighbor nodes and gives higher chances to better opportune nodes for forwarding packets towards a destination. In this way BRP can exploit rich (path) spatial diversity. BRP also implicitly uses multiple paths that circumvent the impact of local instability, failure and mobility.

1.2 Contributions of This Dissertation

The contribution of this dissertation is to enhance security and resiliency over wired networks and to enhance resiliency over wireless networks. To enhance security over wired networks, we focus on preserving critical Internet resources such as critical Internet services and the bandwidth for those services from unauthorized accesses. We mainly preserve critical Internet resources by detecting and filtering out unauthorized accesses including unwanted traffic destined for those critical resources. For this purpose we develop two systems: secure name service (SNS) and lightweight Internet permit system (LIPS). To enhance resiliency over wireless networks, we concentrate on main characteristics of wireless networks and utilize those characteristics to improve resiliency. We develop a scalable and opportunistic bubble routing protocol (BRP) for this purpose. Research results derived from this dissertation have been or will be published in [18, 26, 25, 19, 27, 20]. We summarize the contributions of each development in the following sections.

1.2.1 Secure Name Service (SNS)

We develop the secure name service (SNS) that protects critical Internet resources from unauthorized accesses, denial of service (DoS) and other attacks. The key idea is to enforce *packet-origin authentication* through *resource virtualization.* We realize the resource virtualization through dynamic name binding such that only trusted domains can obtain virtual resource IDs and access those resources with the obtained virtual IDs. In addition these virtual IDs are utilized for packet origin authentication since only legitimate packets can carry security authenticators including virtual IDs. Furthermore dynamic name binding system can effectively protect critical servers under attack by dynamically changing its virtual ID at the service level.

Different from static network-level security schemes such as IPsec and VPN, SNS is able to *dynamically* bind the names of critical resources at the service level, which allows us to actively protect the service resources through a distributed filtering mechanism built on authenticated packet forwarding paths.

We have developed whole framework of SNS and have further addressed the performance and scalability issues in the authenticated packet forwarding. We have also implemented the prototype of SNS on Linux and have shown the feasibility of implementing SNS on regular Linux machines. Finally we have designed fast secure-handle translation schemes to address the scalability issue in fast address translation in security gateways.

1.2.2 Lightweight Internet Permit System (LIPS)

LIPS provides a lightweight, scalable packet authentication mechanism for ensuring traffic-origin accountability. 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 if it accepts or discards the packet. LIPS first introduces *traffic-origin accountability* into the Internet and enables destinations to deny accesses and stop unwanted traffic from untrusted hosts. Second and perhaps 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 the intended targets before launching attacks. Clearly, since a source must obtain a valid access permit before sending packets to a destination, illegitimate/spoofed packets will be automatically filtered out. LIPS has been developed in two modes (i.e., host mode and gateway mode). Host mode is for an end-to-end authentication scheme in which the destination host issues access permits and verifies them, while gateway mode is for domain-to-domain authentication where the permit server issues access permits and security gateways verify them. With gateway mode LIPS helps us to build active defense schemes that automatically identify and fix zombies in LIPS domains through collaboration. When we deploy LIPS as a domain-to-domain approach (i.e., gateway mode) to stop unwanted packets, we do not require broad changes in backbone networks as other approaches. Therefore, LIPS is incrementally deployable in a large scale on common platforms with minor software patches.

In summary, LIPS is designed to localize IP spoofing and associated attacks, restrict worm spreading, stop random probing and reflection attacks, assist intrusion detection systems (IDSs) in significantly reducing their load and providing cross-domain feedbacks, and protect important servers and their incoming links.

1.2.3 Bubble Routing Protocol (BRP)

Wireless networks have unique characteristics of error-prone unstable links and broadcast in nature. We develop a scalable and opportunistic bubble routing protocol (BRP) which enhances resiliency of wireless communications on error-prone unstable links by maximizing the broadcast benefit in wireless networks. BRP neither uses a single or multiple pre-selected paths nor keeps track of next hop information in order to construct paths to the destination as most wireless routing protocols. Instead broadcast is used without designating next hop to deliver control messages or data packets. Therefore no global network topology is needed in BRP. This delivery scheme using broadcast without designating next hop (e.g., unicast) utilizes all neighbors instead of only one neighbor to progress packets and exploits rich (path) spatial diversity. Exploitation of rich spatial diversity circumvents the impact of temporal link/node failures or instability since it implicitly utilizes all probabilistic multiple paths. It therefore improves the end-to-end packet delivery ratio. In addition BRP let better opportune nodes * have higher priority for forwarding the received packets. Furthermore nodes independently make routing decisions without knowledge of global network topology in distributed manner. We also devise a novel greedy set selection (GSS) algorithm which significantly reduces the redundant duplications in control plane and results in the dramatic reduction of the control message overhead. Extensive analysis and simulations show that BRP is most effective in order to reduce the control message overhead and increase end-to-end packet delivery ratio.

* "Better opportune" nodes are closer to the destination or can deliver control messages to the larger number of additional nodes.

1.3 Organization

This dissertation is organized as follows. We first address security issue over wired networks in Part One. We develop the secure name service (SNS) to preserve the critical Internet services from unauthorized access in Chapter 2. We also devise a dynamic directory scheme to support dynamic table management at security gateways. In Chapter 3 we present the motivation behind our approach in constructing a lightweight Internet permit system (LIPS), and the complete design and implementation of LIPS. In addition we extensively evaluate the performance of LIPS through analytical models and real implementation on Linux system. In Part Two, we address the resiliency issue over wireless networks. In Chapter 4 we develop a scalable and opportunistic bubble routing protocol (BRP) that enables highly resilient routing on error prone unstable wireless links over wireless networks. We devise a novel greedy set selection (GSS) algorithm to significantly reduce the redundant control messages. Furthermore we measure the performances through extensive analytical models and simulations. We conclude this dissertation in Chapter 5.

Part I

Enhancing Security over Wired Networks

Chapter 2

SNS: Secure Name Service

2.1 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*, namely, those services that provide significant values.

In this chapter we propose a novel approach – *Secure Name Service (SNS)* – to protect critical Internet services from cyber attacks. 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 and an extension of the standard domain name service (DNS). The basic ideas behind the SNS approach are as follows: A critical Internet service and its associated resources (e.g., servers, databases, etc.) are placed within a (virtual) *secure zone* in the network domain of the service provider, and correspondingly the names of the service and its resources are placed within a *secure name space*, separate from the standard domain name space.

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

In this chapter we describe the proposed SNS architecture which is comprised of two major mechanisms: i) *secure name service* that consists of secure name servers that virtualize protected resources within secure zones, set up security associations (SAs) between domains, and perform secure name resolutions; and ii) *authenticated packet forwarding* that consists of *security checkpoints (SCs)*, *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 *rapid response* mechanisms into our proposed architecture for further securing critical services.

We 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 [52], the SNS framework addresses *active* defense schemes to defeat different attacks at the network level via packet filtering and adaptive forwarding paths. In particular, the dynamic name binding of SNS allows us to take advantage of multiple routes in SNS domains to ensure the availability of services when even some security components are clogged by attacking traffic. Different from previous attack prevention schemes such as SOS [40] and Onion Routing [65] that require relative large-scale infrastructures, the SNS framework can be incrementally deployed in domain by domain.

SNS has its limitations. First, it focuses on packet authentication. Since most applications in collaborative environments employ various security techniques (e.g., TLS [24]) to address the issues of data integrity, confidentiality, and non-repudiation, we emphasize packet authentication to prevent 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 [52] for this purpose.

In the next section we first present the architecture and components of the SNS framework. In Section 2.3 we describe the design of SNS naming scheme. We present the design of authenticated packet forwarding components and the prototype of these components and our experimental evaluation in Section 2.4. In Section 2.5, we devise two fast lookup schemes for secure name translation and evaluate their performance through analysis, simulation and experiments. We then discuss related work in Section 2.6 and summarize the contribution of this work in Section 2.7.

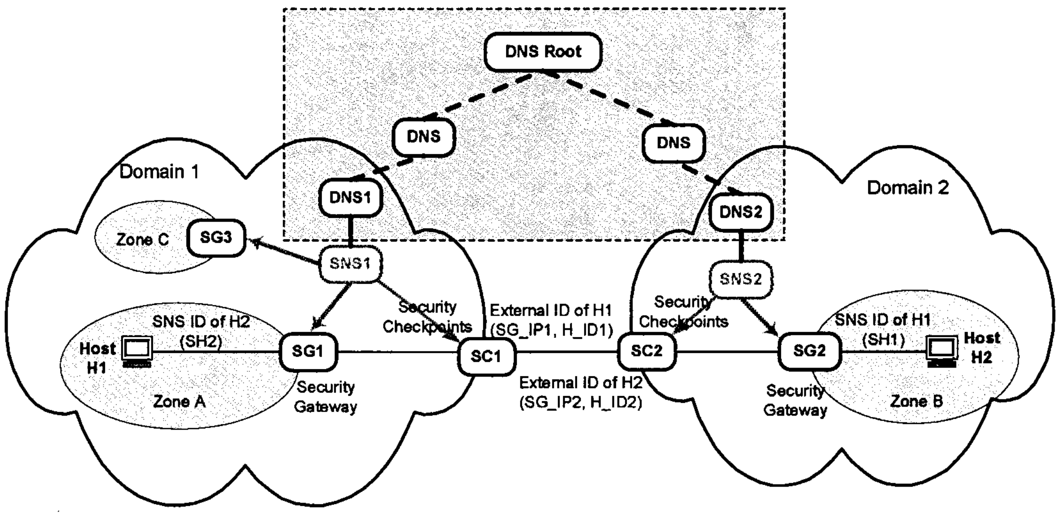


Figure 2.1. SNS Framework

2.2 SNS Architecture and Components

To protect critical resources from unauthorized accesses and DoS attacks, we need multiple levels of security mechanisms to defeat different security threats, e.g., packet replay, flooding, or IP spoofing. In this section, we describe the SNS framework as the first-line of defense to filter out invalid packets and actively protect critical resources. Fig.2.1 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.2.1, 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 to 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.

2.3 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. 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.

2.3.1 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.

2.3.2 Components of SNS Naming System

SNS Server

SNS servers are the key components in the SNS naming framework, which perform in the control functionality 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 2.3.3.

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 [70]; 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

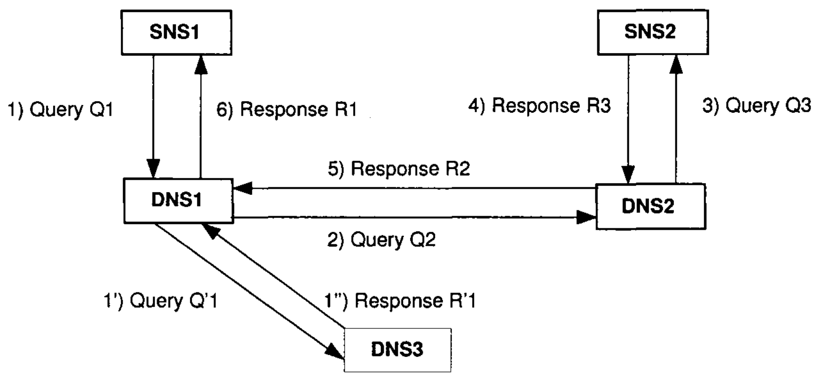
We extend a leaf DNS server into an SNS-aware DNS server such that the SNS service can utilize the DNS service as a bootstrap point for basic naming service. Utilizing the recursive service at leaf DNS servers, SNS servers are able to exchange security parameters for building inter-domain SAs without revealing their IP addresses. As shown in Fig.2.1, 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 [49] 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.2.2. 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"}

Figure 2.2. Process of Exchanging Security Parameters between two SNSs.

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 required in the current DNS and applications. A stub resolver obtains a secure handle (SH) of its SNS from a Secure IP layer* 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.

Secure Handle (SH) Manager

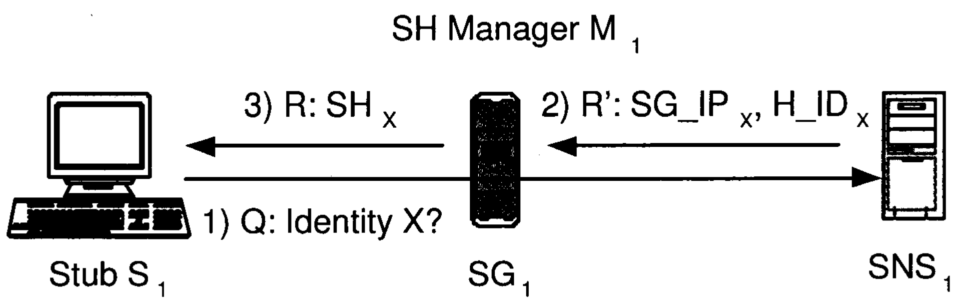
A secure handle (SH) manager maintains an SH database for all secure names at an SG such that the authenticated packet forwarding mechanism at the SG is able to map SHs to their external identities or IP addresses of local hosts in secure name translations. We use a cache-only SH database at an SG, which 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 2.5.

2.3.3 SNS Name Resolution

A secure name resolution maps a secure name into an SNS identity (i.e., SH). The basic process of resolving a secure name query is shown in Fig.2.3.

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

*A secure IP layer is a component of the SNS authenticated packet forwarding mechanism introduced in Section 2.4, which is configured with the SH of an SNS.



SH Database

SNS Identity	External Identity
SH$_X$	SG_IP$_X$, H_ID$_X$

Secure Name Database

Name	External Identity
X	SG_IP$_X$, H_ID$_X$

Figure 2.3. Resolving a Query via Local SNS

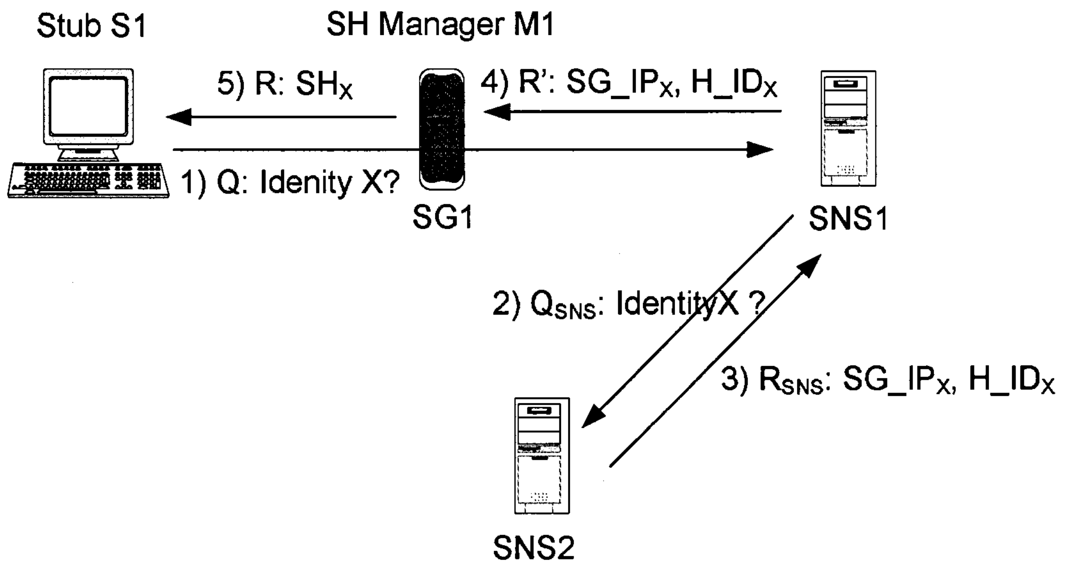

Stub S1
SH Manager M1
5) R: SH$_X$
4) R': SG_IP$_X$, H_ID$_X$
1) Q: Idenity X?
SG1
SNS1
2) Q$_{SNS}$: IdentityX ?
3) R$_{SNS}$: SG_IP$_X$, H_ID$_X$
SNS2

Figure 2.4. Resolving a Query across Domains

SG_1, SG_1 authenticates this message and then forwards it to SNS_1. SNS_1 looks up its secure name database and finds the external identity of X, i.e., (SG_IP_X, H_ID_X). (If X is not in the database, SNS_1 will obtain the external identity of X by issuing a secure name query to SNS server SNS_2 that manages secure name X as depicted in Fig.2.4.) Then SNS_1 passes the external identity of X to SH manager M_1 at SG_1 in a response R'. Upon receiving R', M_1 first checks if the external identity of X is in its SH database. If it is, M_1 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.

2.4 Authenticated Packet Forwarding

2.4.1 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.2.5, 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 destined for 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 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

[†]We assume it is fairly easy to share secret within a domain.

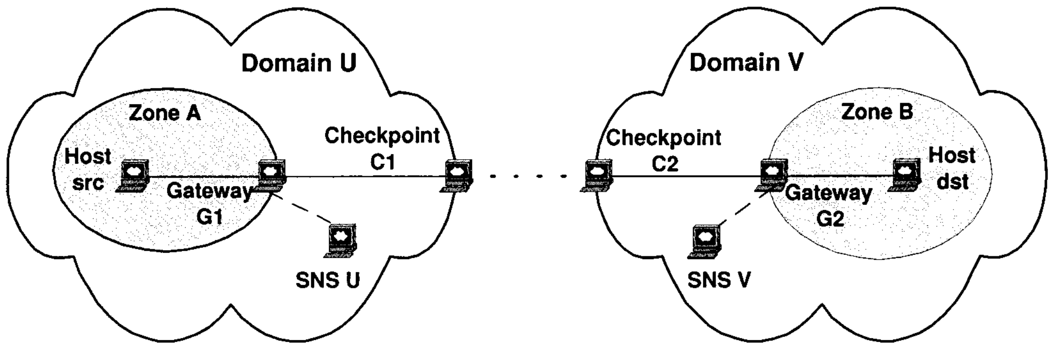


Figure 2.5. Two SNS-enabled Domains

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.

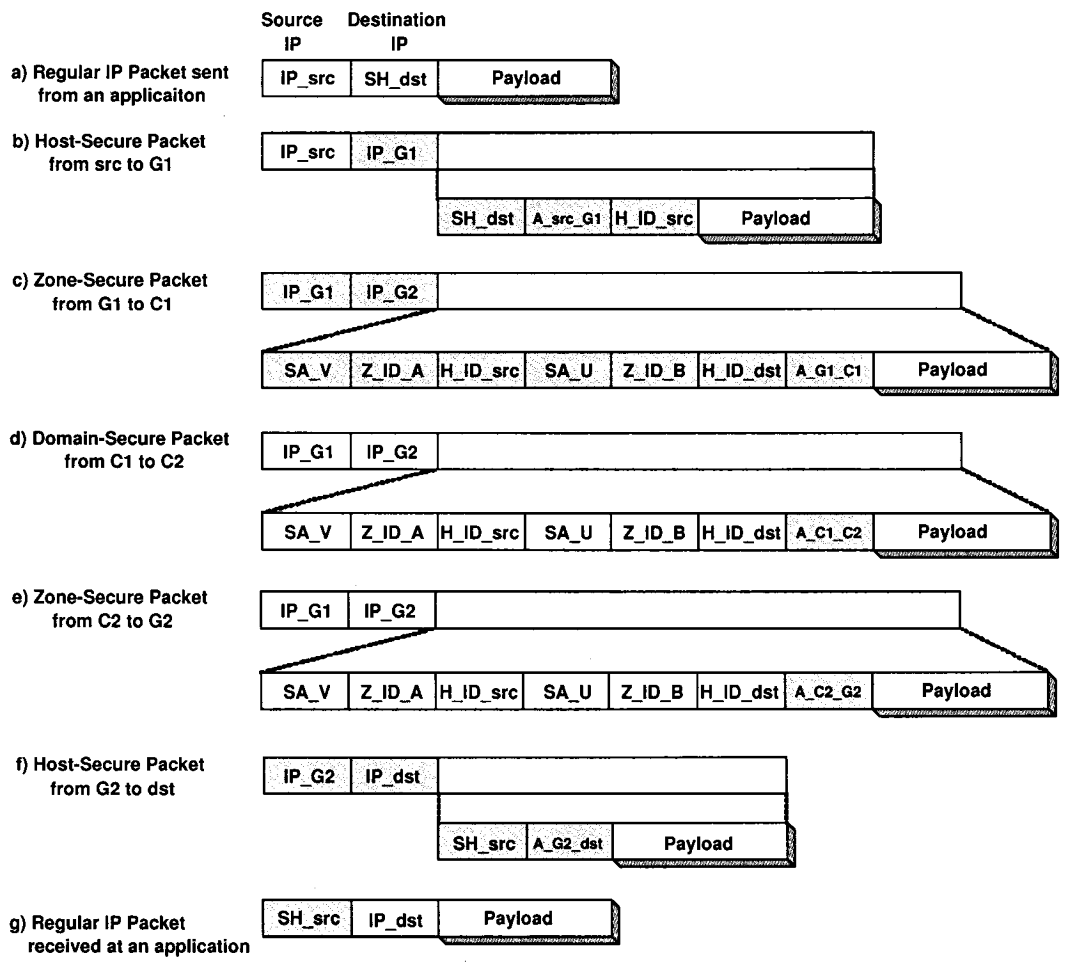
An SC authenticates ingress or egress secure packets. Using Border Gateway Protocol (BGP [66]) announcements, we can control the routes of egress/ingress packets to be routed to chosen checkpoints. An egress zone-secure packet is forwarded from an SG G to an SC C, and it has a zone authenticator generated 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' detects 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.2.5 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.2.6.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.2.6.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.2.6.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.2.6.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



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

Figure 2.6. Packet Formats at each Steps from Host *src* to Host *dst*

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.2.6.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.2.6.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.2.6.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.2.6.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 2.4.2, 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 2.5.

2.4.2 Prototype and Experimental Evaluation

We have implemented the prototypes of sIP layer, SG and SC in Linux kernel 2.4.20 using Linux Netfilter [3] for evaluating SNS authenticated packet forwarding. We refer readers to [18] for the details of the implementation and present the performance results in the following.

We insert a 32-byte packet authenticator (as shown in Fig.2.7) 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 2.8 shows the flow chart of the sIP layer. When an sIP layer intercepts a regular outgoing 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 choose Class E addresses as SHs in the prototype implementation.) 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*

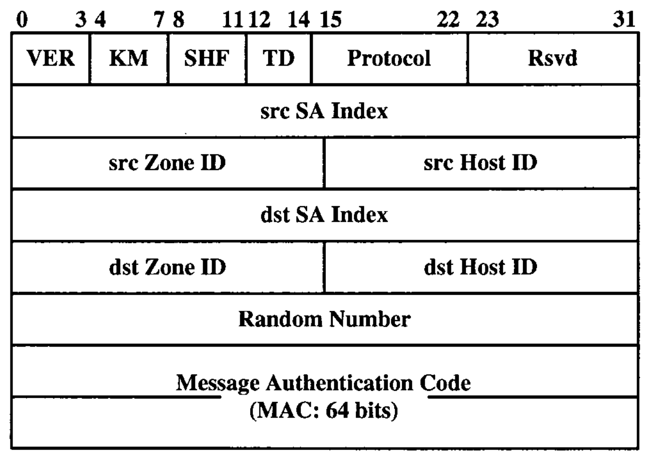


Figure 2.7. Format of Authenticator

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 138. (138 is not assigned to any protocol by IANA yet and we use 138 as the SNS protocol number in the prototype implementation.) A 32-bit random number is also added into the authenticator. After the sIP layer initializes the authenticator, it generates a MAC through *HMAC-MD5* [55] with entries in the authenticator and its host key.

When the sIP layer intercepts an incoming IP packet at the LOCAL_IN hook of Netfilter, it identifies the packet as an SNS packet if the protocol number is 138 in IP header. The sIP layer then generates a testing MAC through HMAC-MD5 with the entries in the authenticator and a host key. If the testing MAC is the same as the MAC in the security authenticator, the packet is valid and the sIP layer replaces

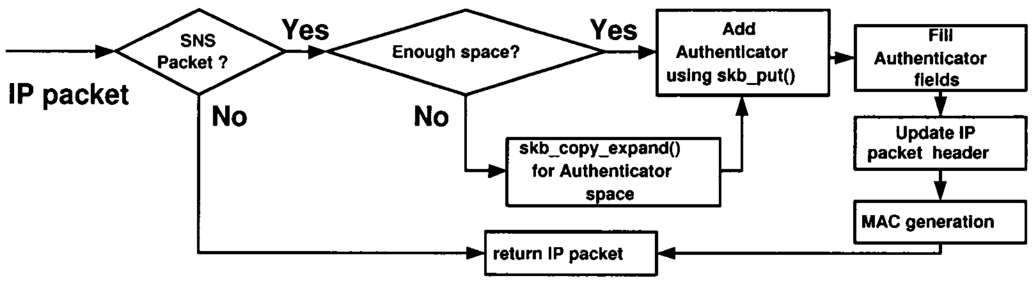

IP packet
SNS Packet ?
Yes
No
Enough space?
Yes
No
Add Authenticator using skb_put()
Fill Authenticator fields
Update IP packet header
MAC generation
skb_copy_expand() for Authenticator space
return IP packet

Figure 2.8. Flow Chart of sIP Procedure

Table 2.1. Delays of MAC Generation

(in clock cycles)

	With precomputed key schedules	Without precomputed key schedules
Blowfish-CBC-MAC	2328	129342
AES-CBS-MAC	2738	12708
HMAC-MD5	-	3812
HMAC-SHA1	-	8368

the source IP address in the IP header with the SH in the authenticator. The sIP layer also restores the protocol field in IP header from the security authenticator into the IP header. Then the sIP layer removes the authenticator of the packet using *skb_trim()* and passes the packet to the IP layer for the further processing.

SGs intercept SNS packets at the PRE_ROUTING and POST_ROUTING hooks of Netfilter for secure packet translations. When an SG receives a host- or zone-secure packet, it retrieves the corresponding key, generates a testing MAC using the key. If the testing MAC is the same as the MAC in the authenticator of the packet, the packet is authenticated, and the SG translates it into a corresponding zone- or host-secure packet. In addition SG adds a source SA index, a destination SA index, a destination zone identifier and a destination host identifier into the authenticator. These parameters are obtained from the secure handle table based on the *SH* of a packet, which is carried in the authenticator of a host-secure packet. The destination

Table 2.2. Delays of Forwarding Components

(in clock cycles)

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

IP address is replaced with the destination SG IP address for the zone-secure packet. The destination host IP address replaces the destination IP address of the host-secure packet. After the SG translates the packet, it generates a new MAC and replaces the MAC in authenticator. The MAC generation is the same as the MAC generation in the sIP layer. When SG intercepts an SNS packet at POST_ROUTING, it replaces the source IP address with its IP address. The main functions of an SC are packet authentication and new MAC generation, which are similar to SGs.

Using the time stamp counter (TSC) of Pentium CPU 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 publicly available codes from NIST, OpenSSL [4] and IETF. As shown in Table 2.1, 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 [69] and AES-CBC-MAC [8] take fewer cycles than HMAC-MD5 and HMAC-SHA1 [5] 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 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

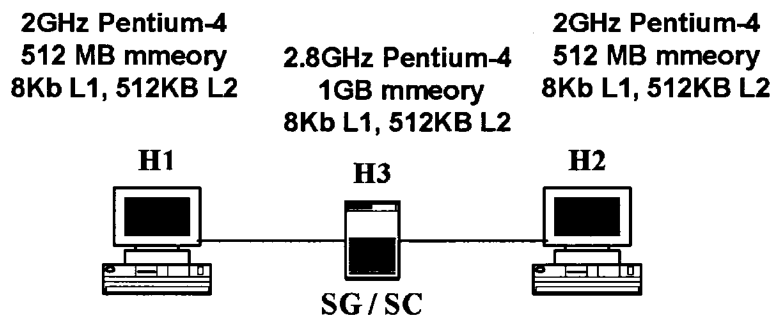


Figure 2.9. Experimental Setting

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 prototype implementation.

We summarize the delays (in clock cycles) at the components of authenticated packet forwarding in Table2.2. For the stress test of sIP layer, we send a large number of UDP packets of 1024 bytes over a direct link between host H1 and H2. We measure the delays of authenticator initialization and MAC generation for each packet at H1. The initialization of an authenticator takes 3067 clock cycles as shown in the first row, including a sk_buff allocation and an IP header update. The MAC generation costs around 3812 cycles. The overall delay of the sIP layer is 6879 cycles (3.44μs). We also measure the effect of sIP on end-to-end bandwidth using *Iperf* [72]. On a 100Mbps link, we achieve a raw transmission rate of 93.9 Mbps in regular IP and a raw transmission rate of 91.9Mbps over sIP, which is 98% of the rate using regular IP.

For the stress test of an SG, we measure the delays of packet authentication, secure packet translation, and MAC generation, as shown in the second row of Table 2.2. We connect host H1 to H2 through H3, which acts as an SG, as shown in Figure 2.9. Again, we send a large number of UDP packets of 1024 bytes from H1 to H2. We use the similar setting of SG to test H3 as an SC. The results are also shown in the third row of Table 2.2. The last column in Table 2.2 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 of sIP layer, SG and SC have shown the feasibility of constructing SNS using regular PCs for common LANs.

2.5 Dynamic Table Management at SGs

The secure handle (SH) lookup mechanism at SGs is critical to its performance and scalability because it needs to lookup an SH for each packet from a potentially large address table. Therefore, we focus on this issue in this section.

2.5.1 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 to directly 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 multi-level directory scheme is shown in Figure 2.10, where each directory entry is corresponding to a unique address table entry. The single-level directory scheme is shown in Figure 2.11, 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 section. 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

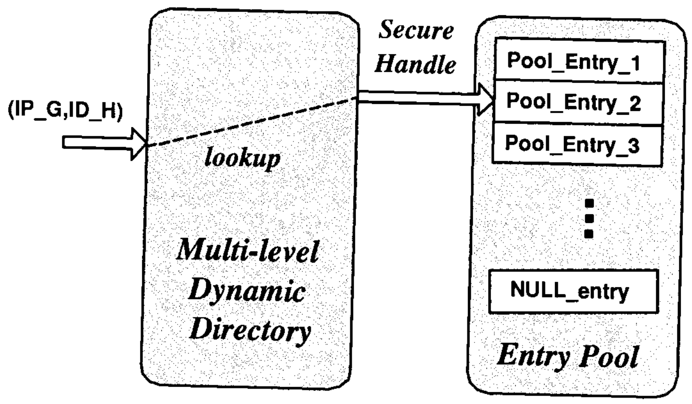


Figure 2.10. Multi-Level Directory

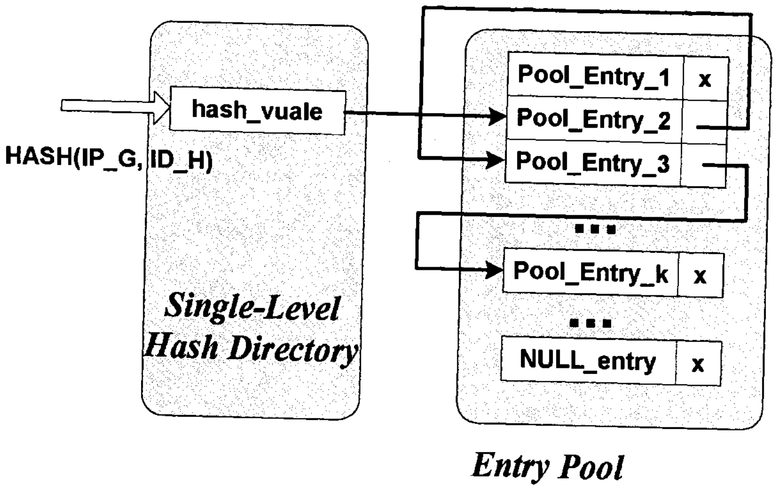


Figure 2.11. Single-Level Directory

uses this SH for packet forwarding between a local host and itself.

2.5.2 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 [22], 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.

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. Algorithm 1 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 $N_l = \frac{N_1}{2^{16+8\cdot(l-2)}}$, $2 \leq l \leq 5$.

Algorithm 1 Lookup Algorithm of Multi-Level Directory

```
1: if entry $e$ is empty then
2:    INSERT($i$) // insert client $i$ into entry $e$
3:    return a secure handle
4: else
5:    if exact one client is in entry $e$ then
6:       if $i$ is the same as the client in entry $e$ then
7:          return a secure handle
8:       else
9:          collision occurs
10:         EXPAND() //expand a next level directory
11:         INSERT($i$), INSERT($i'$) // insert both into the next level
12:         return a secure handle
13:      end if
14:   else
15:      // at least two clients are in entry $e$
16:      step down into the next level directory
17:   end if
18: end if
```

Algorithm 2 Lookup Algorithm using Linear Hashing

```
1: if $H_i(key) \geq p$ then
2:    index = $H_i(key)$
3: else
4:    index = $H_{i+1}(key)$
5: end if
6: access the entry at the index
7: search through a overflow list if necessary
```

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 2.1.

$$\begin{aligned} D_{insert}^l &= d_{flag} + p_0^l \cdot d_{insert} + p_1^l[d_{compare} + d_{expand} \\ &\quad + E_{insert}^{l+1}(i,i')] + p_2^l[d_{down} + D_{insert}^{l+1}] \end{aligned} \tag{2.1}$$

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 2.2.

$$E_{insert}^l(i,i') = \tfrac{1}{2^{16+8\cdot(l-1)}} E_{insert}^{l+1}(i,i') \;\; +(1-\frac{1}{2^{16+8\cdot(l-1)}})\cdot 2\cdot d_{insert} \tag{2.2}$$

where $2 \le l \le 4$. 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^l_{lookup}, is given recursively by Equation 2.3.

$$D^l_{lookup} = d_{flag} + p^l_1 \cdot d_{compare} + p^l_2[d_{down} + D^{l+1}_{lookup}] \tag{2.3}$$

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

$$D(i) = P^{(i)}_{insert} \cdot D^1_{insert} + (1 - P^{(i)}_{insert})D^1_{lookup} \tag{2.4}$$

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 $F = 1$), $1 \le l \le 4$. The probability that flow i is collided with another entry at level l is $m(i,l) = (\prod_{k=1}^{l-1} p^k_2) \cdot p^l_1$. 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 $2^{k-1} \cdot \tilde{N}_0 \le N_1 \le 2^k \cdot \tilde{N}_0$. 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 Algorithm 2.

$$D_{lookup} = d_{hash} + d_p + D_{list} \tag{2.5}$$

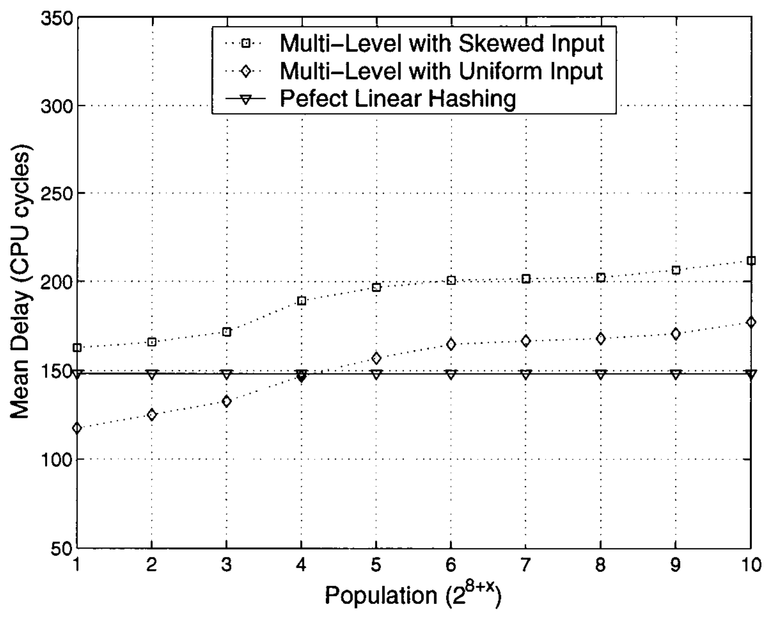
where d_{hash} is the delay of computing the hashing function, d_p is the delay to compare with a splitting pointer p, and D_{list} is the expected delay of searching through the overflow list. For a good hashing function, we assume that the average length of the list is less than two. As a result, the 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

$$D_{insert} = p_0 \cdot d_{insert} + p_1 \cdot (d_{hash} + d_p + D_{list} + d_{insert}) \tag{2.6}$$

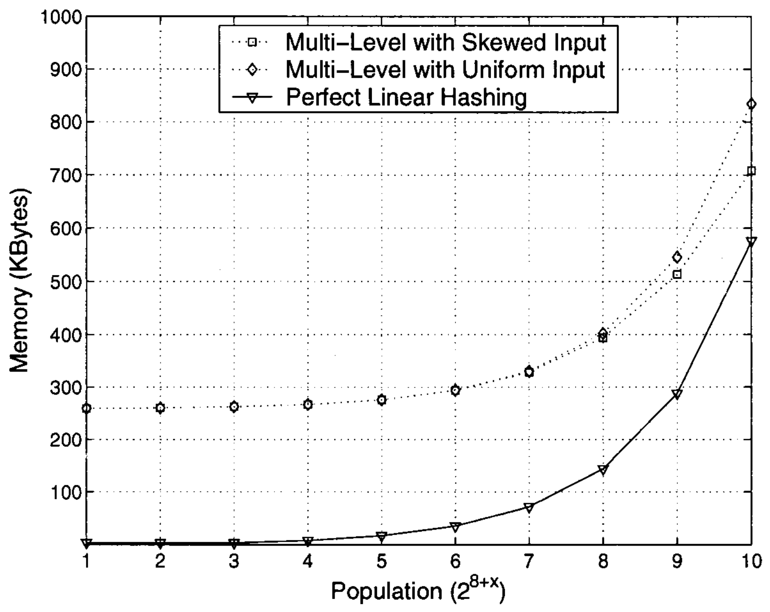
And the expected lookup/insertion delay of packets of flow i is

$$D(i) = P^{(i)}_{insert} \cdot D_{insert} + (1 - P^{(i)}_{insert}) \cdot D_{lookup} \tag{2.7}$$

We measure the delay of memory read/write and hashing computation in Linux kernel and plug in these parameters into our models. Fig.2.12 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

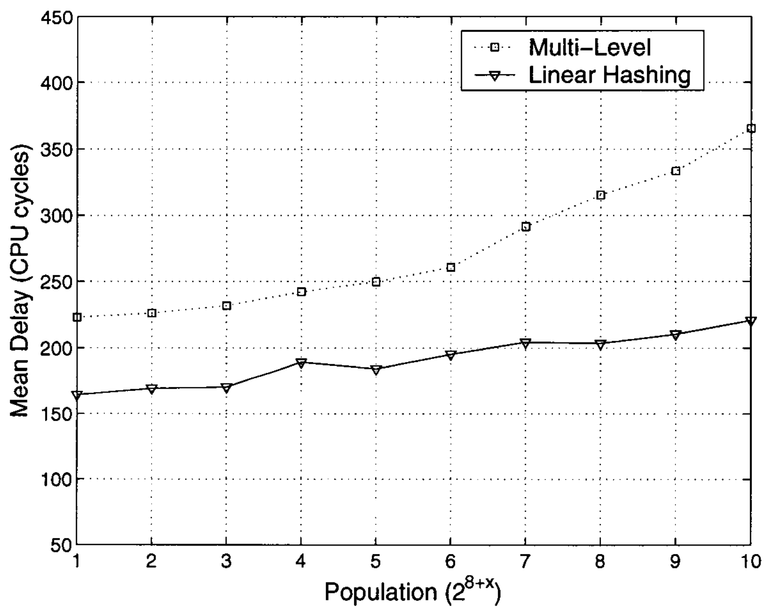


(a) Mean Delay

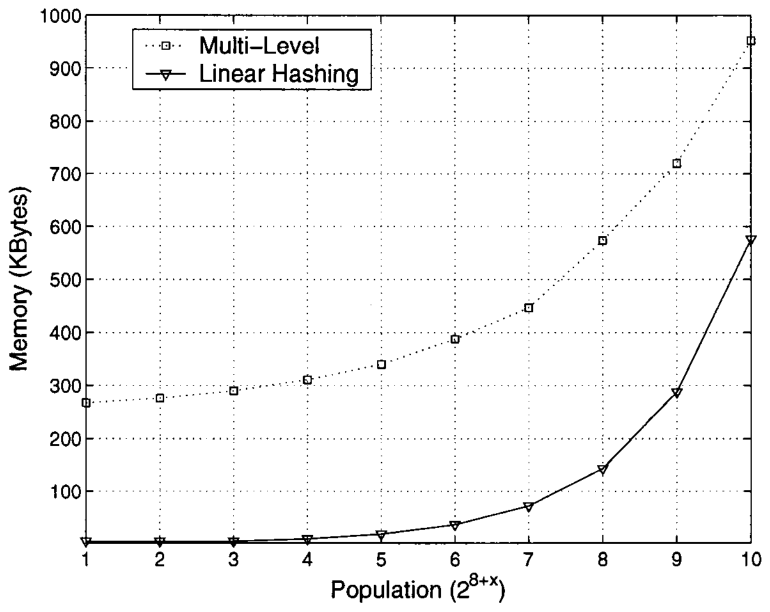


(b) Memory Cost

Figure 2.12. Comparison of Delay and Memory Cost with Analytical Models



(a) Mean Delay



(b) Memory Cost

Figure 2.13. Comparison of Delay and Memory Cost using Simulations

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.2.12.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.2.13.a shows the mean delay of the hashing scheme is significantly better than the multi-level scheme. Fig.2.13.b shows that the memory cost of the hashing scheme is also better than the multi-level scheme.

2.6 Related Work and Discussion

In SNS, we combine name service and network security into a unified framework. In this section, we review the related work in naming security, traffic security, entity authentication, and proactive and reactive defense schemes. For naming security, DNSSEC [11, 12] 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 [39,35] or L2TP [73] are common approaches used to address the traffic security issue for extranets. TLS [24] ensures the security at the transport layer. Kerberos [52] 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 [30], Secure Overlay Service (SOS) [40], Mayday [9], and VPN Shield [63]. 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 [68], link testing, input debugging [71], controlled flooding [16], logging [71], ICMP trace-back [1], packet marking [16, 68], 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.

Compared to the related work, the proposed SNS shows several salient advantages. First, the SNS framework provides a comprehensive first-line of defense through resource virtualization and dynamic name binding, which allows us to apply different security policies at multiple levels and components to address different security threats. As a result, it enhances the service availability with low management costs. In particular, SNS distributes the security check load over SGs and SCs in authenticated packet forwarding, and therefore significantly reduces the security loads at critical servers. SCs are responsible for filtering out ingress attacking traffic, while SGs mostly emphasize secure-packet translation. Consequently, critical servers can sustain their service performance under attacks. In contrast, existing approaches such as VPN or TLS could not address this issue. As a result, a critical server could not sustain its performance under attacks because it has to devote itself to intensive security checking. Furthermore, SNS is incrementally deployable as it does not require to have a broad infrastructure in place, and it does not require to replace application software. In addition, SNS not only is capable of providing real-time responses to attacks, but also greatly reduces the administrative burden of securing communications with automatic secure name resolutions. No operators are involved in real-time operations, such as resolving secure names, automatically setting up rules for ingress/egress filters, and isolating ill-behaved hosts. Besides, since SNS blocks invalid packets and monitors traffic at security gateways and checkpoints, more sophisticated intrusion detection systems can focus on packets that reach critical resources to further improve security. Moreover, SNS not only actively protects these hosts from unauthorized attacks from outside, but also monitors and stops any attacks started by insiders. Likewise, the dynamic name binding of SNS also well suits to support the name resolving of mobile hosts.

2.7 Summary

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.

Chapter 3

LIPS: Lightweight Internet Permit System

3.1 Introduction

The success of the Internet can be attributed to its simple design philosophies: *hour-glass* protocol stack, *open* architecture, and *end-to-end* principle, among others [21]. The *decentralized* and *open* nature of the Internet enables innovative technologies, services and applications to be created, deployed and spread rapidly. Implicit in the original design of the Internet is *open trust* – 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 kinds of cyber attacks, while enjoying the comfort of not being able to be tracked down easily. As the number of network security attacks with the aim to abuse or disrupt Internet services has significantly increased every year, the sophistication of these attacks has also been sharply escalated. Massive distributed denial-of-service (DDoS) attacks are such an example. Consequently, cyber threats have been serious issues in today's Internet. In enhancing the current Internet with security mechanisms, and evolving it into a more secure infrastructure, important trade-offs such as openness of the Internet, efficacy and flexibility of security protection, cost of deployment and ease of use must be carefully studied and evaluated.

The key security issue in the current Internet is that 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), spreading attacks to many other vulnerable hosts. To combat this problem, many organizations choose to "close off" their networks via mechanisms such as VPNs [39, 63, 35], or employ firewalls to block certain types of packets (e.g., based on IP addresses, ports, or packet payload), regardless of senders and their intent. Clearly, such solutions are fairly limited in their scope or effectiveness as email viruses and worms 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. [21].

In this chapter, we propose a novel *lightweight Internet permit system (LIPS)* to provide traffic accountability through fast packet authentication for stopping unwanted packets. By unwanted packets, we mean packets *not* intended for "normal" communications between hosts, such as packets with spoofed IP addresses, generated in port scanning or worm spreading. Such packets account for, or are forerunners of, most of unauthorized accesses, intrusion, disruption, denial-of-service (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 an efficient traffic authentication mechanism to filter out most of these illegitimate packets, with minor changes to current systems and negligible overheads. We implement LIPS as a small patch to the IP layer at a LIPS-aware host. 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 *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 into LIPS, we can build an effective and scalable "first-line" defense to protect Internet resources from unwanted traffic.

The remainder of this chapter is organized as follows. In Section 3.2, we present the basic concepts and constructs of LIPS, and illustrate how it works. In Section 3.3, we present the design and implementation of LIPS. First simple host mode in LIPS is presented and then we present the LIPS gateway mode for enhancing system performance and scalability. In Section 3.4, we evaluate the performance of LIPS via simulations and experiments on our prototype implementation on Linux platforms and demonstrate its efficacy in stopping unwanted traffic with negligible overheads. We discuss the related work in Section 3.5 and conclude this chapter in Section 3.6.

3.2 Basics of LIPS Architecture

The idea of LIPS is simple but very efficient: 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 to send packets to a destination, it must obtain a valid access permit first. 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. As only data packets with valid access permits will be accepted and processed by applications running 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.

3.2.1 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.3.1. Protocol type 138 is not yet assigned by IANA and we choose 138 as the protocol type of LIPS in our prototype implementation. To avoid the potential segmentation issue, we first perform a path maximum transmission unit (MTU) discovery and then set a proper MTU for the connection. The format of a LIPS packet header is given in Fig.3.2, 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 is 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

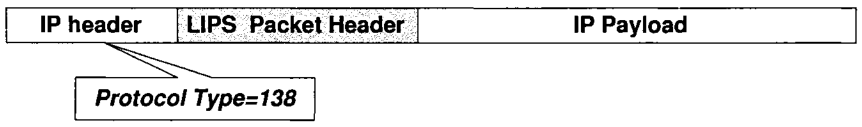


Figure 3.1. Converting an IP Packet into a LIPS Packet.

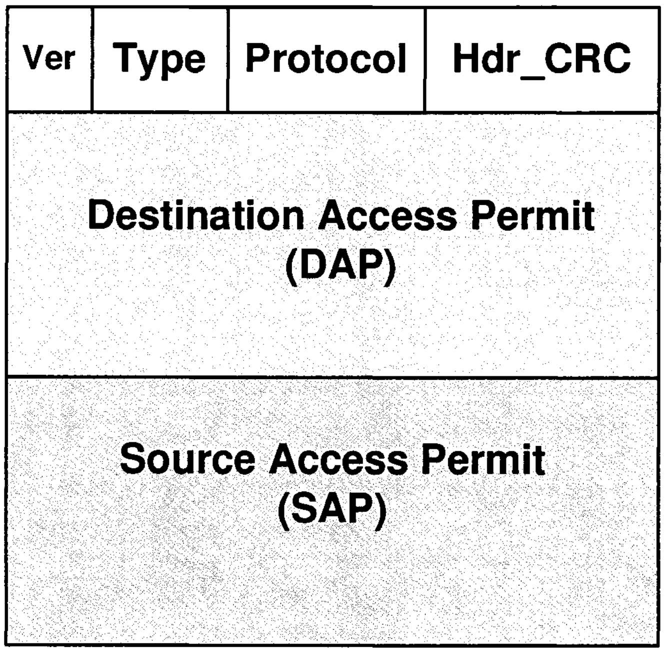


Figure 3.2. Format of LIPS Packet Header.

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 in 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 Cyclic Redundancy Check (CRC) for a LIPS packet header.

3.2.2 Access Permit

An access permit is constructed using *keyed* message authentication code (MAC) [47] at a LIPS-aware host. This MAC is generated through a secure hash function with

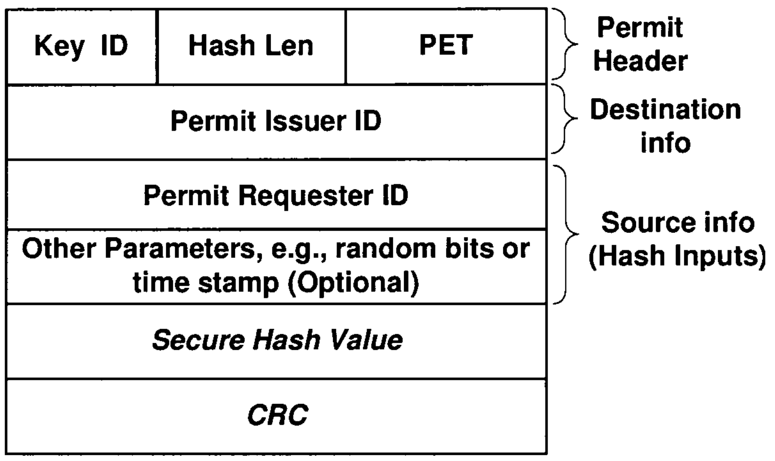


Figure 3.3. Format of LIPS Permit.

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.3.3, 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 and several optional security parameters (e.g., a random number used to deal with permit-replay attacks). It 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 [55], 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

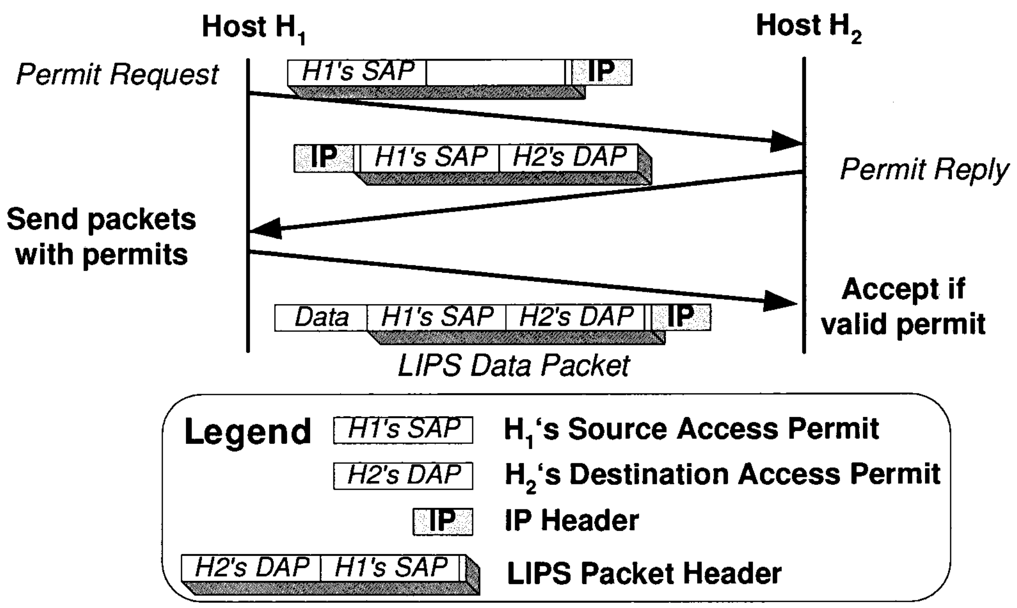


Figure 3.4. LIPS Message Exchanges for Setting up Access Permit.

permit header is l, the secure hash value of a permit is the first l bits of the 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 certain period of time, as K_t will be changed over time. Without knowing the secret key, it is very difficult to forge a permit.

3.2.3 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.3.4, 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. As shown in the figure, this request carries H_1's SAP in the request. 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.

An example policy may be only accepting permit requests from a local domain, e.g., accepting packets from (secured) proxy servers when communicating with hosts in other domains. This will automatically eliminate port scanning and worm spreading packets from other domains, i.e., an attacker outside a domain cannot discover vulnerabilities via scanning and a worm cannot propagate across domains through random probing. In this way damages can be localized. 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 request from H_1) as the DAP. This simple example shows the case where a LIPS-aware host directly exchanges permits with another in a LIPS host mode. We will further introduce domain-level permit exchanges in Section 3.3.2 when we introduce the LIPS gateway mode.

H_1 will only accept a permit reply that carries a valid DAP, namely, a SAP issued by it 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*. Otherwise the packet is discarded. For the subsequent data packets sent to H_2, H_1 puts 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.

3.2.4 Permit Cache

Each LIPS host maintains a permit cache with a format shown as Table 3.1. A cache entry contains a destination IP address, a *Flag*, and a destination access permit. For

Table 3.1. Permit Cache

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

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.

3.2.5 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, e.g., IPsec [39]. Hence the overhead of key management in LIPS is minimal.

*Although in the LIPS gateway mode (see section 3.3.2) 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.

3.3 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 introduced in the previous section. To further improve its security strength and capability, we 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 deployed in a large scale with light permit traffic and short inter-zone permit-setup delays.

3.3.1 LIPS Host Mode

The LIPS host mode is used as an incremental approach to deploy LIPS when a few 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 deploy 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 the HAL include exchanging permits, maintaining a permit cache, attaching permits to packets, and verifying access permits, as will be explained below. We implemented HAL in Linux 2.4 kernel using Linux *Netfilter* [3]. The HAL uses 256 128-bit secret keys and employs HMAC-MD5 [55] as the secure hash function. As reported in Section 3.4, our *software* implementation on a common Linux platform can achieve a high packet authenticating 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_recv()* 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 depending on configured security policies. If it is a LIPS permit request/reply, the HAL executes the permit setup protocol as follows.

Permit Setup Protocol

To obtain a permit for a destination, the HAL sends a permit request message to the destination, creates/updates a cache entry for it 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 message to the requester as introduced in Section 3.2. When the HAL at the requester receives this reply, it first verifies its DAP. If the DAP is valid, it searches through its permit waiting queue and processes all packets destined for 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 a permit reply is lost, a HAL may mistake a LIPS-aware host for 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. For incremental deployment, 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 retransmits the permit request message to the destination. If still no responses at the third timeout, the HAL treats the destination as a non-LIPS host and set the flag to 2 in the cache entry.

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 [45], 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.

The LIPS host mode has several limitations. First, it is subject to flooding attacks that directly hit a LIPS host. Furthermore, it is not suitable for large scale deployment due to periodical permit refreshes (required for security purposes), potential long delays in permit maintenance (due to distance), and a large number of permits to maintain and packets to exchange. Therefore, we develop the LIPS gateway mode to address these issues.

3.3.2 Design of LIPS Gateway Mode

To address the scalability and security issues in the 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 communication 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 to 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 routers and gateways, attackers have very little chances to sniff zone permits between them. Therefore, permit-mutation *localizes damage* caused by potential attacks as most attacks are started from compromised end hosts. We will discuss this issue more in the following. In each zone, LIPS-aware hosts still directly communicate with each other as in the LIPS host mode.

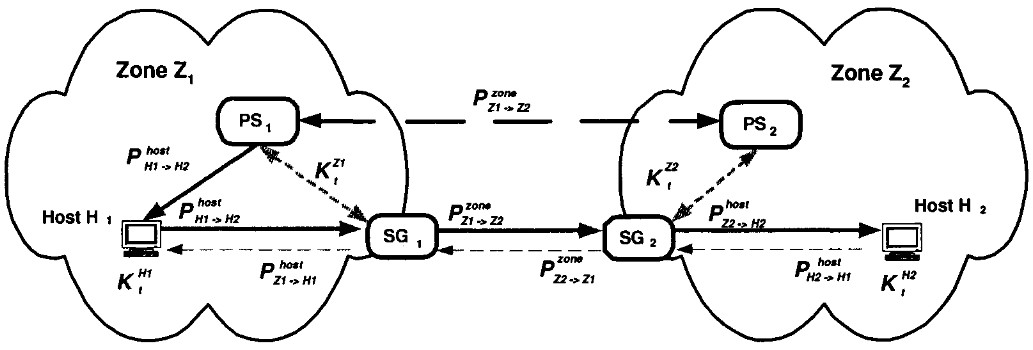


Figure 3.5. Illustration of LIPS Gateway Mode

Permit Server, Intra-zone and Inter-zone Permit Setup Protocol

As show in Fig.3.5, host H_1 in zone Z_1 wants to access host H_2 (e.g., a protected 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. Table 3.2 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 will be exchanged between PSs. We use this zone ID to generate a zone access permit as

Table 3.2. Summary of Notations

Notation	Definition
$P_{x \to y}^{host}$	Host Access Permit issued to host x by host y
$P_{x \to y}^{zone}$	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

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. 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.

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

†For a PS to find an authoritative PS of a 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.

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 $P_{Z_1 \to Z_2}^{zone} = H(IP_{PS_1}, K_t^{Z_2})$, where $H()$ is a secure hash function, and the CRC checksum of the permit is computed on $P_{Z_1 \to Z_2}^{zone}$. 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 $P_{Z_1 \to Z_2}^{zone}$, the requesting PS *mutates* it into a host permit $P_{H_1 \to H_2}^{host}$ 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, $P_{H_1 \to H_2}^{host} = P_{Z_1 \to Z_2}^{zone} \oplus H(IP_1, IP_2, K_t^{Z_1})$. Note that the host permit $P_{H_1 \to H_2}^{host}$ 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 $P_{Z_1 \to Z_2}^{zone}$ replaced by $P_{H_1 \to H_2}^{host}$. 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

[‡]For ease of exposition, we will also refer to the secure hash value contained in an access permit as simply the access permit.

differently. During its initialization, a HAL authenticates itself to its permit server and security gateways, e.g., via a Kerberos [52] system. 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 the permit 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, $P^{zone}_{Z_2 \to Z_1} := H(IP_{PS_2}, K_t^{Z_1})$, 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.3.5. 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, $P^{host}_{Z_1 \to H_1}$, which it uses to authenticate packets from PS_1 and SG_1 back H_1, where $P^{host}_{Z_1 \to H_1} := H(IP_{PS_1}, K^{H_1}_t)$, IP_{PS_1} is the IP address of PS_1, and $K^{H_1}_t$ is a secret key of H_1. $K^{H_1}_t$ is periodically refreshed. Separate access permits 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.

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 $P^{zone}_{Z_1 \to Z_2} = H(IP_{PS_1}, K^{Z_2}_t)$, and $K^{Z_2}_t$ 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, $P^{host}_{H_1 \to H_2} := P^{zone}_{Z_1 \to Z_2} \oplus H(IP_1, IP_2, K^{Z_1}_t)$, where $K^{Z_1}_t$ 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 a *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 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.

3.3.3 Advantages of LIPS Design and Implementation

The design and implementation of LIPS have the following several salient 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. [§] Packet authentication is performed using only information carried in (the LIPS header of) a packet and *secret keys* held *locally* by security gateways and hosts. Thus packet operations can

[§]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".

be done efficiently – our initial experimental testing in Section 3.4 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/host maintains access permits for destination hosts with which it is currently communicating. The size of such caches are in general 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 any *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* modification to applications is required. Through separate *zone-level* and *host-level* access permits, we isolate "bad" packets originating in its own zone from those outside, and limit the abilities of attacks to mostly "man-in-the-middle" *replay* attacks by "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. The "man-in-the-middle" replay attacks between domains are in general much difficult to launch, as border gateway routers are typically connected via high-speed fiber optical links, and they are significantly harder to gain access to. Permit- or Packet-Replay attacks can be mitigated by including, e.g., sequence no. or random bits, in access permits. 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 very *localized* effect, with only "sniffed" hosts/domains being affected, due to the host-specific/domain-specific feature of access permits.

3.4 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.

3.4.1 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 [55]. 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 [37], 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 Linux PC. With such a high permit 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 defeat 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 [7], our LIPS implementation can authenticate traffic at a rate around 640 Mbps on a common Linux PC, far beyond a common user's requirement.

Table 3.3. 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

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 3.3 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 3.3.1, we use a linear hashing scheme with controlled splitting to manage a permit cache. The initial cache size is 1 MB with 2^{16} entries. We choose a control threshold of 0.75 as recommended in [45]. 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 traffic traces (UMN and WorldCup) in our simulations as the input of a permit cache 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 [6], including 73 millions web requests and more than two millions different destinations. As summarized in Table 3.4, our permit cache management schemes performs reasonably well under both theoretical models and real traces.

Table 3.4. 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 3.5. 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

Overall Overhead of LIPS

We conduct experiments to examine the overall overhead introduced by our LIPS implementation in data transmission, compared with IP. In these experiments, we use Iperf [72] to send CBR UDP traffic 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 3.5 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%.

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 flooding attacks 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. 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. 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.

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 [30], 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 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 no legitimate traffic is sent 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.3.6, 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.

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.3.7 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.3.8 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.3.9 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 protect incoming link of a target [40]. 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

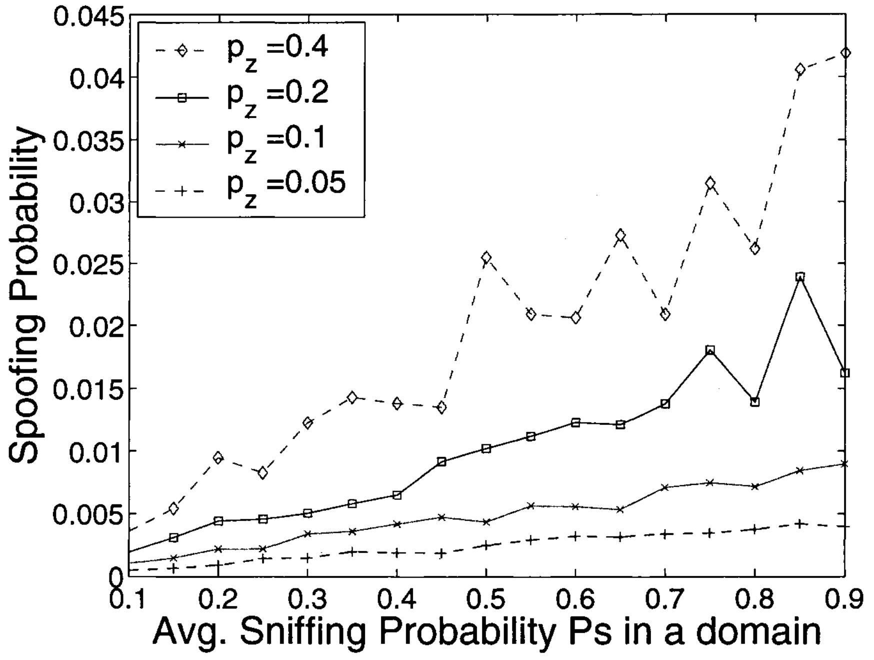


Figure 3.6. Spoofing Probabilities in LIPS for a Zombie

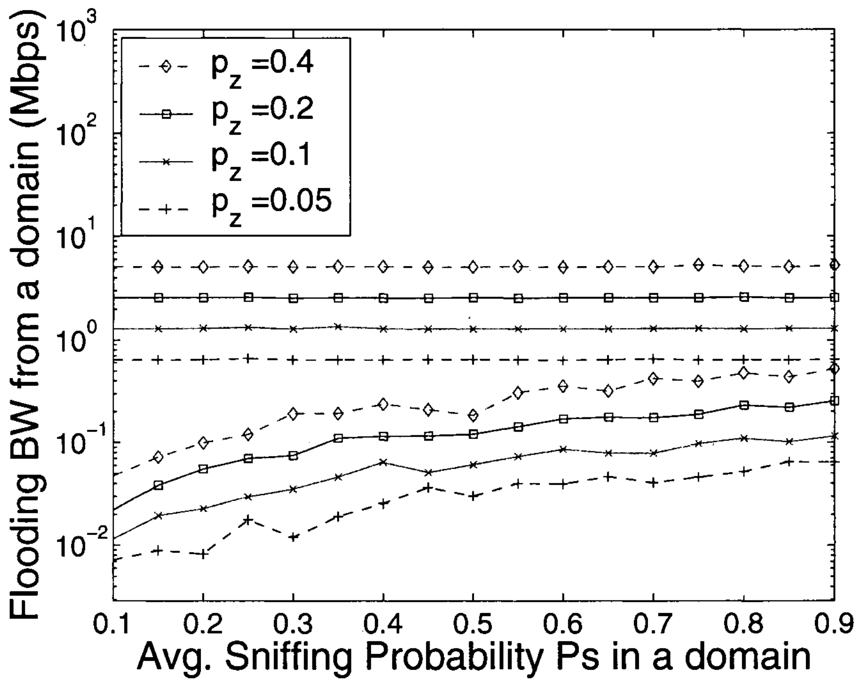


Figure 3.7. Aggregate Flooding Bandwidth:IP with Ingress/Egress Filtering vs. LIPS

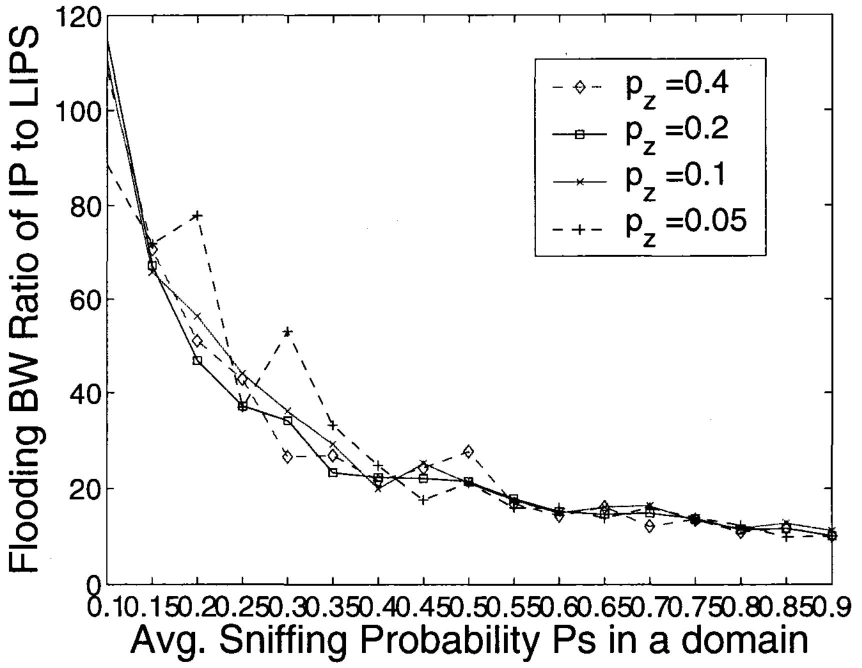


Figure 3.8. Potential Flooding Bandwidth Ratio of IP with Ingress/Egress Filtering to LIPS

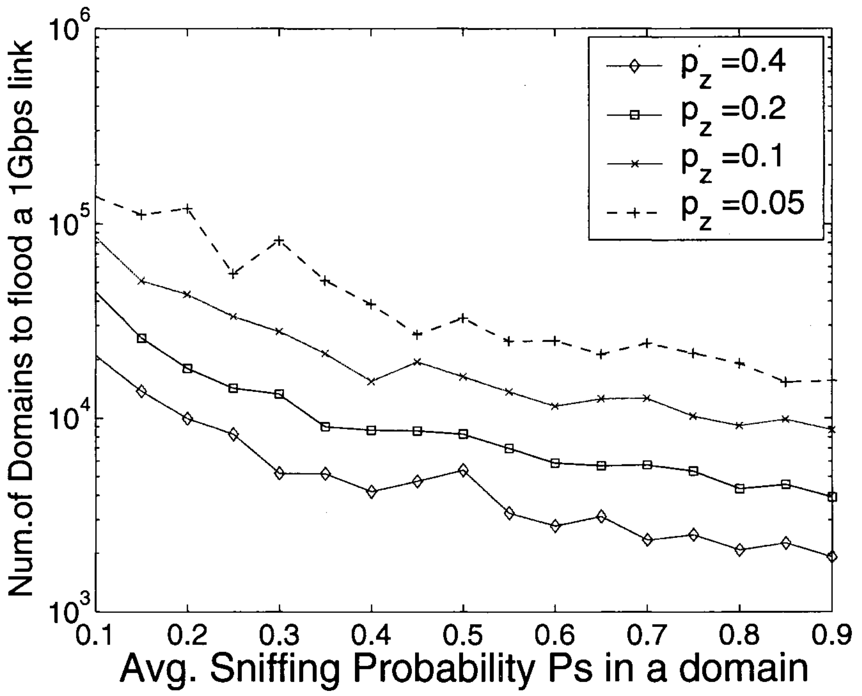


Figure 3.9. Number of Domains Needed to Flood a 1 Gbps Destination Link

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 the legitimate users [40]. The load level of attack traffic has to be significantly higher than that of legitimate traffic in order to blocking legitimate traffic. Fig.3.10 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.

Since zombies have to generate very high attacking load as shown in the above to launch successful attacks, it is very easy to identify them and isolate them. Therefore, we can easily detect these zombies at source domains through a traffic monitoring scheme and then isolate these zombies through a local defense mechanism, e.g., instantly reconfigure filters at routers to drop all packets from these zombies and take further actions later using more advanced approaches. Furthermore, we can also detect flooding attacks at a destination domain and inform source domains to take care of corresponding zombies through inter-domain collaboration. In this case, we first detect a flood at the 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. Furthermore, once the source PS confirms this event, the PS revokes the host permit at its SG(s), automatically drops outgoing packets with the host permit, and instantly reconfigures filters at a local router of the zombie to drop packets form 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. 3.11 and Fig. 3.12, we can quickly shut off these replaying source in 10 seconds, with $D_f = 0.1$ second.

3.4.3 Stopping Flooding Attacks from non-LIPS Domains

Here we use a simple experiment setting to show how the LIPS protects a link from being flooded by traffic from non-LIPS domains. The experimental setting is shown in Fig.3.13. 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 non-LIPS attacking traffic. We set the capacity of H_2's incoming link to 2 Mbps using Linux class based queueing

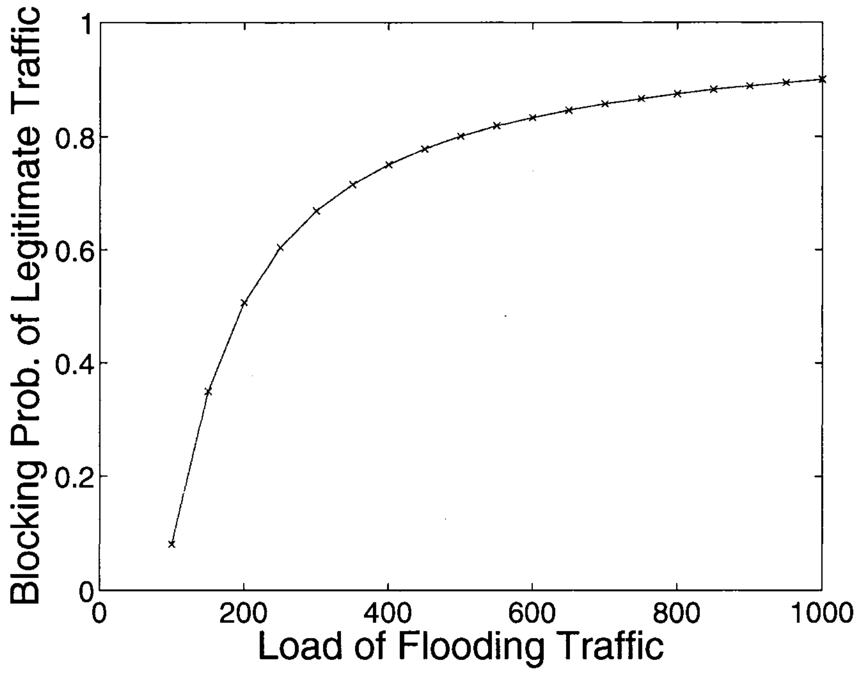


Figure 3.10. Blocking Probability of Legitimate Flows as Attacking Traffic Load increases

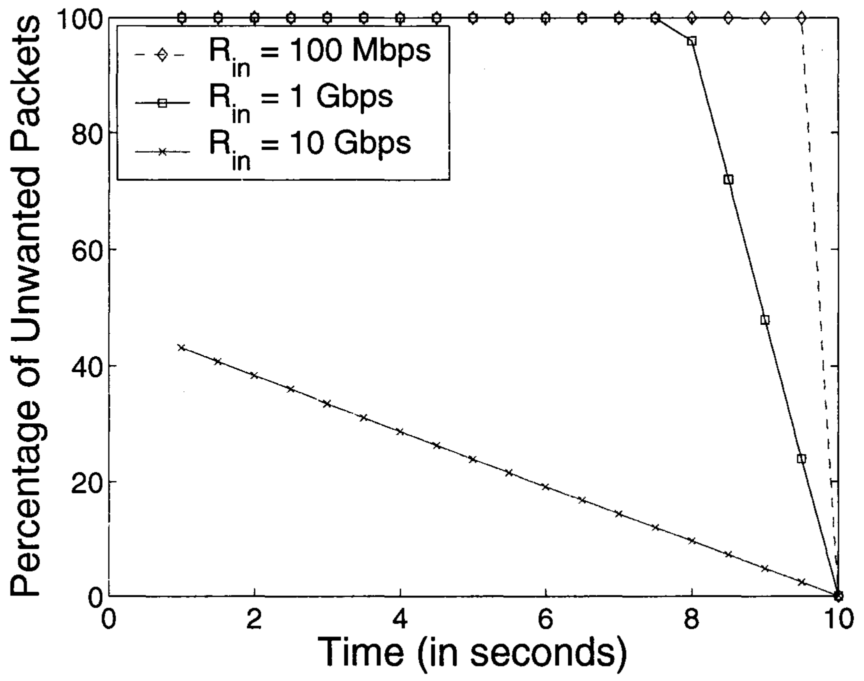


Figure 3.11. 1,000 Replaying Sources: 100 Sources in each of 10 Domains

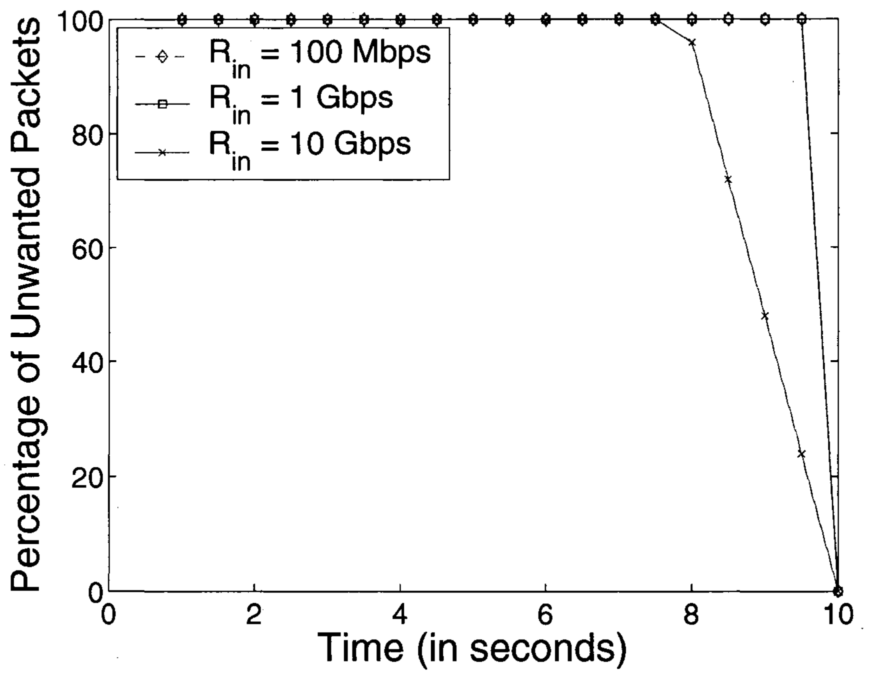


Figure 3.12. 10,000 Replaying Sources: 100 Sources in each of 100 Domains

(CBQ) [2] method. 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 reaching the H_2's incoming link is lower than its capacity, the attack causes almost no damage on F_l with or without LIPS. However, when the total traffic rate is higher than the link 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 3.6, Table 3.7, and Table 3.8, respectively. On the contrary, when we use the LIPS gateway mode to protect the link, flow F_l reaches H_2 at its required bandwidth with no packet losses and negligible jitters, as shown in the third super-columns in the tables.

3.4.4 Security Strength of Access Permits

A successful attack on access permits is equivalent to a birthday attack, in which attackers look for two messages (e.g., M and M'), that produce the same secure hash

Table 3.6. Measured Bandwidth

(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 3.7. Measured Packet Loss

(Percentile)

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

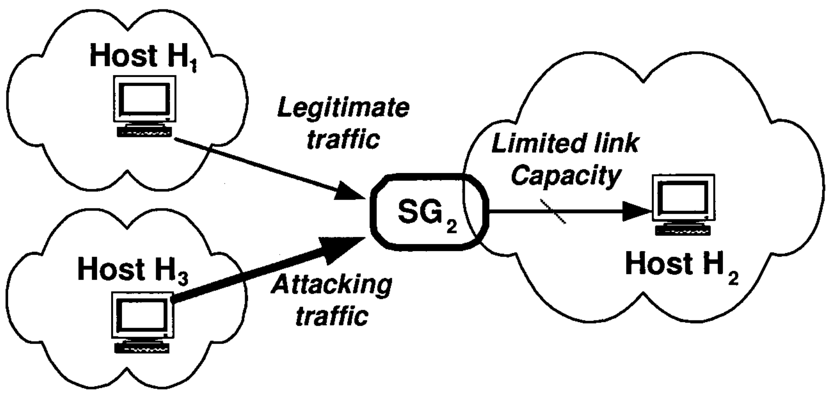


Figure 3.13. Preventing Flooding Attacks from non-LIPS Domains

Table 3.8. Packet Jitter

(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

Table 3.9. Lower Bound for Birthday Attack on Permits

Link Rate	64-bit Hash Value	96-bit Hash Value
100Mbps	800 days	143616 years
1Gbps	80 days	14362 years

value (e.g., $H(M) = H(M')$. Because attackers do not know the secret key, they need to observe a sequence of hash values generated with the same secure hash key in order to conduct a successful attack [47].

Table 3.9 shows the lower bound for attackers to collect sufficient data on a 100Mbps or a 1 Gbps link in order to break LIPS permits. Clearly, the probability of successful attacks on permits is rather small, especially when a host randomly uses a key from its *periodically-refreshed* 256-key pool.

3.5 Related Work

Many security mechanisms have been proposed to control IP spoofing and the damage of associated distributed denial of service (DDoS) attacks, or trace back attacking sources. Ingress filtering is a static approach for preventing spoofed IP addresses outside its domain, and it is most effective for stub network domains and less so for *transit* domains [30]. Furthermore, ingress filtering does not preclude an attacker using a forged source address within a legitimate prefix range.

Both host identity protocol (HIP) [51] and Statistically-Unique-and-Cryptographically-Verifiable (SUCV) identifiers [50] use public-key schemes at a host level to address the spoofing issue. However, the computational overhead of public key schemes limits host performance and makes them difficult to scale to large systems.

Spoofing Prevention Method (SPM) [14] attaches a domain key to each packet and verifies the key at the destination domain, similar to our zone permit introduced in Section 3.3.2. However, similar to ingress filtering, SPM does not address the issue of spoofing in a valid address space.

Pushback [56] treats DDoS as a congestion-control problem and requires each router to detect and drop packets that probably belong to an attack. Upstream routers are also notified to drop such packets in order that the router resources are used to route legitimate traffic. The network capability scheme [10] inserts special tokens into packets and use 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. Furthermore, overlay approaches (SOS and Mayday [40, 9]) use a *wide-area* overlay infrastructure with a *large number* of intermediate nodes to filter out attacking traffic. Moreover, hop-integrity protocols [31] provide secure communications between adjacent routers by computing a message digest for *each packet* at *each forwarding step*. IP Easy-Pass [75] aims to protect real-time priority traffic from Denial-of-QoS attacks at an ISP edge router by maintaining *per-flow states*. In addition, the visa protocols [29] use encryption and data signatures to authenticate a flow of packets. They require a shared key to be established between access control servers on a *per-source-destination* basis. Similarly, IPsec and VPNs establish shared keys to secure end-to-end communications with known overheads [34, 48]. Several traceback schemes were proposed to track down attacking sources, such as IP traceback [68]. These schemes usually require to modify intermediate routers along packet forwarding paths and are mostly for post-attack analysis. SAVE [43] propagates valid source addresses between intermediate routers on forwarding paths.

In summary, these approaches generally either incur high computational overheads and/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 not adopted in a large scale. On the contrary, the proposed LIPS does not use encryption or digital signatures, hence the overheads of encryption and key management are minimized. Furthermore, LIPS is an end-to-end/edge-to-edge approach that neither requires any support from *intermediate* networks nor need to modify any intermediate routers. Therefore it is easier to be deployed incrementally in a large scale.

3.6 Summary

We conclude this chapter by summarizing the contributions of LIPS. LIPS enforces preliminary traffic-origin accountability and enables destination hosts or domains with the ability to deny access and stop unwanted traffic from unauthorized sources. Since a source node must obtain an access permit to a destination before sending traffic, the illegitimate spoofed traffic is automatically filtered out. Furthermore we can better defend against worm spreading and denial of service (DoS) attacks by simply *detecting anomalies in the permit request traffic.* For example, a sudden and unusual surge in the number of permit requests to one or more hosts in a protected network signifies suspicious activities. In particular, when implemented in the gateway mode, a source zone can detect attacks originating from malicious or worm/virus-affected hosts within its zone and quarantine them by denying (host-specific) permit requests to the target destinations. Hence combined with network intrusion mechanisms, LIPS can form an effective first line of defense against cyber attacks by stopping unwanted traffic.

In summary, LIPS prevents and localizes IP spoofing and associated attacks. In addition enforcing traffic-origin accountability in LIPS can easily stop worm spreading, random probing, and reflection attacks.

Part II

Enhancing Resiliency over Wireless Networks

Chapter 4

BRP: Bubble Routing Protocol without Graphs

4.1 Introduction

Most multihop wireless routing protocols have been adopted or variated from wired network routing protocols and are use of *pre-selected* single or multiple minimum cost paths. [60, 38, 61, 57, 41, 74, 42, 46, 36, 13] They use conventional graph theory to pre-select paths. Links are represented by edges and a path by a sequence of edges in the conventional graph theory.

However, unlike wired networks, variable and unpredictable capacity of wireless links makes routes highly unstable and failure-prone in wireless networks. This instability and failure-prone characteristics of wireless links makes adoption of existing routing protocols in wired networks and the usage of the conventional graph theory inadequate for wireless networks. Likewise the employment of pre-selected static path(s) on failure prone links hinders resilient communications. In addition wireless networks broadcast in nature. This broadcast nature can exploit rich (path) spatial diversity that is not applied for most existing routing protocols in wireless networks.

In this chapter we propose a scalable and opportunistic *bubble routing protocol* (*BRP*) that neither uses a single or multiple pre-selected static path(s) nor tracks next hop information. Therefore BRP does not utilize conventional graph theory. Instead BRP employs an opportunistic forwarding scheme in the delivery of control messages and data packets. Upon receiving control messages or data packets, receivers await a small amount of backoff time (e.g., this backoff time is independently estimated based on the number of uncovered neighbors or a route metric for a destination at each node) and forward the packet (*wait-and-forward*). More opportune nodes, which are closer to the destination or can deliver control messages to the larger number of extra nodes, have smaller backoff time and better chances to forward the packets than less opportune nodes. When a node forwards control messages or data packets, the node broadcasts them instead of designating a next hop and receivers decide further forwarding of packets towards a destination. In this way BRP can exploit rich (path) spatial diversity and implicitly use multiple paths that circumvent the impact of local instability, failure and mobility. In addition each node listens to channel and suppresses received packets if they have been further progressed towards the destination by better opportune nodes (*listen-and-suppress*). BRP decreases redundant duplications of control messages or data packets through this listen-and-suppress scheme.

In summary the objective of BRP is to provide a resilient routing scheme without causing exchange of control messages in multihop wireless networks. Here "resiliency" means that packets are more reliably delivered to destination through unstable links. Different from most existing routing protocols, BRP does not depend on traditional graph model to discover route metrics and deliver data. Instead BRP utilizes *bubble (area)* for route metric discovery and data delivery while most existing routing protocols use *links*. BRP is composed of following two stages:

- Discovery of route metrics in control plane: We aim to get a route metric for a destination that reflects network topology. Even though BRP gets a route metric for a destination, the path *IS NOT* pre-selected (i.e., no next hop information maintained at each node) and forwarders are decided at the delivery time of control messages or data packets. Only route metric will be obtained through route metric discovery process in control plane. We precisely explain the route metric discovery process in section 4.3.

- Delivery of data in data plane: Once a route metric is obtained for a destination, data are delivered through broadcast and does not follow pre-selected path(s). Data delivery exploits rich spatial diversity and follows the best available routes at the delivery time. Data delivery process is precisely described in section 4.4.

4.1.1 Related Work

Wireless routing has been an active area in networking researches and many wireless routing protocols have been proposed. [38,61,60,57,36,42,41,46,74] These traditional routing protocols have been adopted or modified from wired network routing protocols. Likewise they first *preselect* a minimum cost single path or multiple alternate paths and utilize the preselected static path(s) in transmitting data. For this purpose, each packet carries a full path from the source to the destination in it [38] or each node maintains next hop information. [61,60,57,36,42,41,46,74] At the specific time, these traditional routing protocols use a single path via unicast even though they prepare multiple alternate paths during route discovery process. Multiple alternate paths are used one by one only if the primary path is broken. But bubble routing protocol (BRP) delivers control messages and data packets through broadcast and utilizes as many as neighbors to transmit data. Therefore BRP exploits spatial diversity by simultaneously using local multiple paths right at packet delivery time. BRP neither preselects any single or multiple alternate paths nor maintains next hop information.

Some wireless routing protocols have been proposed to avoid broadcast storm. [53, 33, 58, 62, 59] A predetermined probability [53, 33] or counter [53] is used to reduce redundant rebroadcasting messages at each node. But assigning appropriate predetermined values is non-trivial as network topology or traffic patterns change. Distance-based or location-based scheme [53] is proposed but the overhead of measuring precise location degrades the performance. In addition the chosen link quality may not be consistent with the location information. The basic idea of scalable broadcast algorithm (SBA) [58] is similar to greedy set selection (GSS) algorithm used in BRP in terms of delaying of rebroadcast messages and independently deciding the amount of delay in distributed manner. However SBA utilizes the degree of nodes and the maximum degree of neighbors while GSS gives higher priority of forwarding packets to nodes that can deliver packets to the larger number of additional nodes.

ExOR [13] is influential opportunistic routing protocol for wireless mesh networks. ExOR preselects prioritized forwarding candidates and each packet carries those forwarding candidates list. Only receivers in the forwarding list forward packets in the order of forwarding priority estimated based on the proximity (measured using ETX [23]) to the destination. To reduce redundant transmissions, ExOR utilizes a batch map which records the received packets at each node. ExOR imposes the overhead of selection of prioritized forwarding candidates, and the selection of appropriate forwarder list is not trivial for a source-destination pair which are in multihop range. Westphal has proposed opportunistic routing in dynamic ad hoc networks (OPRAH protocol). [76] OPRAH prepares multiple paths for a destination and each packet carries forwarding node list. Here forwarder list is dynamically adapted to the mobility. These opportunistic routing protocols preselect forwarder list or prepare multiple paths and utilize them while BRP never preselects path(s) and forwarders are decided at the time of data delivery. Furthermore these protocols mainly focuses on data plane improvement with underlying link state routing protocol or AODV-style control plane protocols while BRP simultaneously focuses on the reduction of control message overhead and the enhancement of data delivery ratio.

The rest of the chapter is organized as follows. Section 4.2 presents BRP operation and design concerns. Route metric discovery and data delivery schemes are presented in section 4.3 and 4.4, respectively. Section 4.5 describes a novel greedy set selection (GSS) algorithm that significantly reduces redundant control messages, followed by evaluation of BRP in Section 4.7. We conclude the chapter in section 4.8.

4.2 Bubble Routing Protocol (BRP)

In this section we describe operation of bubble routing protocol (BRP) and design concerns in BRP.

4.2.1 BRP Operation

BRP can operate in proactive, on-demand, and hierarchical routing mode. The main difference of those three modes are the route metric discovery scheme. In proactive mode, route metrics are exchanged among neighbors whenever route metric changes are detected. In on-demand mode, route metric request and reply messages are exchanged whenever nodes need route metrics for destinations. Hierarchical routing mode is the combination of the proactive and on-demand modes. For example proactive mode can be deployed inside a domain and on-demand mode for the outside of domain. In this paper we propose BRP in on-demand mode to increase the scalability. Three control messages are defined in BRP: route metric request (RMREQ), route metric reply (RMREP), and HELLO. HELLO messages are periodically exchanged among neighbors and each node can detect the change of neighbors through HELLO messages. RMREQ and RMREP are crucial for route metric discovery process. When a route metric for a new destination is needed, a node originates a route metric request (RMREQ) message to obtain route metric for the destination. Every node receiving the RMREQ decides if the node can reply back to it, further forward it, or drop it. Upon receiving the RMREQ message a node can reply back to it with a RMREP message if the node is the requested destination or has *fresh enough* * route metric for the destination. The node drops the received RMREQ message when the received RMREQ is a duplicated one or already replied one. More importantly, the

*'Fresh enough' means that the intermediate node has a route metric for the destination whose associated sequence number is greater than one carried in the RMREQ.

node suppresses a RMREQ message if the node's all neighbors receive the RMREQ message. Otherwise the node waits a small amount of time (i.e., backoff time) and forwards the RMREQ message. This backoff time is assigned based on the estimated number of neighbors that may not receive the RMREQ message.

A route metric is determined when the RMREQ message reaches either the destination itself or an intermediate node with fresh enough route metric for the destination. The route metric for the destination is delivered to the origination of the RMREQ through broadcasts of a RMREP message. Like processing RMREQ, each node independently decides if the node drops or forwards the received RMREP message. The node drops the RMREP when the node does not have a route metric for the originator of a corresponding RMREQ or the received RMREP is a duplication. In addition the node drops the received RMREP message if a *better opportune* neighbor [†] for the originator forwards the same RMREP message. Otherwise the node waits a small amount of time and forwards the RMREP message.

Each node receiving RMREQ or RMREP messages caches route metrics for the origination of the RMREQ or for the destination of the RMREP. Generating and processing control messages are described in section 4.3

Data packets are delivered through broadcast and carry route metrics for a destination. BRP transmits data packets in batch mode where a batch is composed of k (e.g., 100) packets. When a node receives a data packet, the node can drop or forward the packet. A node drops data packets whenever it receives duplications of packets or the better opportune neighbor forwards the packet. Otherwise the node forwards the received packets after backing off a small amount of time. The details of delivery of data packets are described in section 4.4.

[†]Better opportune node has a route metric for a destination with the greater destination sequence number or the smaller route metric with the same sequence number.

4.2.2 Design Concerns

Route metric discovery messages are delivered via broadcast in BRP and there are several concerns that have to be carefully addressed. At the early stage of network bootstrap, RMREQ messages must be delivered to every node in the network since the originator of RMREQ and intermediate nodes do not know the direction of a destination. For this purpose controlled flooding scheme (e.g., each node always forwards a RMREQ message once whenever the node receives a new RMREQ message.) can be applied. However controlled flooding could cause many unnecessary rebroadcasts of control messages especially for dense networks. Therefore we must design BRP to reduce *unnecessary* duplications of control messages as much as possible. BRP applies *wait-and-forward* and *listen-and-suppress* schemes to reduce unnecessary duplications. Upon receiving RMREQ message each node waits a small amount of time and forwards the RMREQ message, or suppresses the RMREQ message if all the node's neighbors receive the RMREQ messages during the waiting time. We need to develop a novel scheme to identify how long a node waits before forwarding the RMREQ message and to realize that all neighbors receive the RMREQ message.

In addition to distribution of RMREQ messages, RMREP messages are delivered to the origination of RMREQ through broadcast in BRP. Likewise controlled flooding is inefficient to deliver a RMREP message to the origination of a corresponding RMREQ and we need to develop an efficient delivery scheme that reduces unnecessary duplications and still delivers the best route metric to the origination of the RMREQ.

BRP addresses these concerns in the design of two schemes that reduce unnecessary duplications. The first scheme, greedy set selection (GSS), is devised for the distribution of RMREQ messages. RMREQ messages carries forwarder's neighbor list and whenever a node receives a RMREQ message it marks which neighbors are covered [‡] by the received RMREQ. In addition a node estimates waiting (i.e., backoff) time based on its uncovered neighbor set. Intuition is that the less covered neighbors by the received RMREQ, the smaller waiting time at each node. A node with more uncovered neighbors has higher priority to forward the RMREQ message. We analyze the efficiency of this scheme in section 4.5. The other scheme is for RMREP message and uses the route metric for the origination of RMREQ message at each node. Only better opportune nodes for the origination of a corresponding RMREQ forward the RMREP message. Upon receiving a RMREP message, a node estimates waiting (backoff) time based on route metric for the origination of a corresponding RMREQ message. The smaller route metric a node has for the origination of the corresponding RMREQ message, the higher priority the node has for forwarding the RMREP messages. Furthermore a node suppresses a RMREP message whenever any better opportune node forwards the same RMREP messages before the node forwards the RMREP message. This scheme is also applied for the data delivery in data plane.

[‡]Covered neighbors means the common neighbors between the RMREQ forwarder and the receiver.

4.3 Route Metric Discovery

The route metric discovery process includes the scenarios under which nodes generate route metric request (RMREQ) messages and route metric reply(RMREP) messages and how the messages are processed. In order to correctly process the messages, certain state information (e.g., destination sequence number) must be maintained in the route metric table entries for the destinations of interest.

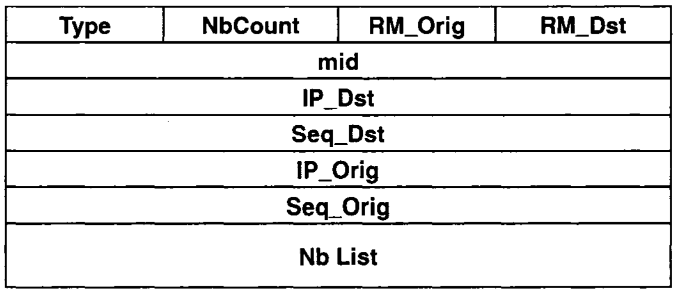


Figure 4.1. Route Metric Discovery Message

As shown in Figure 4.1, route metric discovery messages are composed of control message type (Type), the number of neighbors (NbCount), the route metric towards originator of RMREQ (RM_Orig), the route metric for the queried destination (RM_Dst), the message identifier (mid), the destination IP address (IP_Dst), the destination sequence number (Seq_Dst) for the destination that is known to the originator in the past, the originator IP address (IP_Orig), the destination sequence number (Seq_Orig) for the originator, and the list of neighbors (Nb List). We summarize notations used in the chapter in Table 4.1. In the chapter we use distance vector (e.g., hop count) route metric for the expedition without loss of generality. However BRP can also utilize any route metric (e.g., ETX [23] or ETT [28]) with negligible modification. For example, we can replace the distance vector with the ETX value for destinations and nodes with smaller ETX values have higher priority to forward received RMREP messages or data packets.

Table 4.1. Notations

Notation	Definition
N_i	a set of node i's neighbors
$N_i \setminus N_j$	a set of i's neighbors that are not common in j's neighbors
S^x	Destination sequence number of a node x
D^x	Route metric (e.g., the number of hops) towards a node x
D_i^x	Route metric (e.g., the number of hops) from a node i to a node x
S_i^x	Destination (x) sequence number at a node i

4.3.1 Dissemination of Route Metric Request (RMREQ)

A node disseminates a new RMREQ message when the route metric is not available for a destination of interest. When a node i receives a RMREQ message from a neighbor node j, it processes the RMREQ message as in the Algorithm 3. Upon receiving a RMREQ, a node increments the Orig_RM (e.g., D^s) by one and updates its route metric for the originator of the RMREQ only if the RMREQ carries a "fresh enough" or better route metric for the RMREQ originator. After processing the route metric for the originator of the RMREQ, each node can drop or forward the RMREQ or reply back to the RMREQ originator. A node discards the received RMREQ when any duplicated RMREQ arrives or all neighbor nodes receive the RMREQ message or the node receives a corresponding RMREP message during waiting time. An intermediate node can forward the RMREQ if the node does not have a fresh enough route metric for the destination. Before broadcasting the received RMREQ, a node keeps a message identifier (e.g., *mid*) with the originator IP address (e.g., *IP_Orig*) and through the maintained (*IP_Orig*, *mid*) pair, the node identifies duplication of RMREQs and discards it.

Algorithm 3 Processing RMREQ($s, mid, d, S^d_s, D^s, S^s$) at node i

```
if $i = d$ then
  // $i$ is the queried destination
  // update the originator's RM if needed
  if $S^s_i < S^s$ then
    $S^s_i = S^s$
    $D^s_i = D^s + 1$
  else if $(S^s_i = S^s)\&\&(D^s_i > D^s + 1)$ then
    $D^s_i = D^s + 1$
  end if
  if $req\_state(s, mid) \neq Replied$ then
    broadcast RMREP($s, mid, d, 0, S^d_i, D^s_i, S^s_i$)
    $req\_state(s, mid) = Replied$
  end if
else
  // $i$ is an intermediate node
  // update the originator's RM if needed
  if $S^s_i < S^s$ then
    $S^s_i = S^s$
    $D^s_i = D^s + 1$
  else if $(S^s_i = S^s)\&\&(D^s_i > D^s + 1)$ then
    $D^s_i = D^s + 1$
  end if
  if $req\_state(s, mid) = (Forwarded||Replied)$ then
    drop RMREQ
  else
    // Not Replied or not Forwarded RMREQ
    if $S^d_i > S^d_s$ then
      // $i$ has a fresher route metric and sends RMREP
      broadcast RMREP($s, mid, d, D^d_i, S^d_i, D^s_i, S^s_i$)
      $req\_state(s, mid) = Replied$
    else
      // $S^d_i \leq S^d_s$
      // $i$ has out-dated metric
      $D^s = D^s + 1$
      wait for $T_{backoff}$ period assigned based on $N_i \setminus \mathcal{C}$
      if a neighbor node replied during $T_{backoff}$ then
        drop RMREQ
        //RMREP is processed
      else
        if all neighbors in $N_i$ received the RMREQ then
          drop RMREQ
        else
          broadcast RMREQ($s, mid, d, S^d_s, D^s_i, S^s_i$)
          $req\_state(s, mid) = Forwarded$
        end if
      end if
    end if
  end if
end if
```

4.3.2 Scheduling of RMREQ Messages

Before a node forwards a received RMREQ message, the node waits small amount of time (e.g., $T_{backoff}$) in order to give more chances of forwarding to higher priority nodes (*wait-and-forwarding*). Choosing this waiting time is very challenging since BRP does not employ centralized control. Instead the waiting time should be independently assigned in distributed manner at each node without knowledge of the global network topology. We propose a novel greedy set selection (GSS) algorithm that utilizes neighbor information for assigning waiting time. GSS is precisely described in section 4.5. HELLO messages are exchanged to announce its existence to neighbor nodes and RMREQ messages carry the sender's neighbor list. Upon receiving a RMREQ message, a node (e.g., i) checks which neighbors can potentially receive the RMREQ message and add the neighbors to *covered neighbor set* $\mathcal{C}$. Nodes in $\mathcal{C}$ are common neighbors between the sender and the receiver of the RMREQ. A node updates $\mathcal{C}$ whenever the node receives duplicates of the RMREQ. BRP assigns shorter waiting time as the number of uncovered neighbor set ($|N_i \setminus \mathcal{C}|$) increases. Further neighbors generally have larger number of uncovered neighbors. In this way BRP gives higher priority to nodes that are farther away from the sender of the RMREQ and maximizes the reduction of unnecessary duplications.

4.3.3 Handling Route Metric Reply (RMREP)

When a node receives a RMREQ message, the node generates a route metric reply (RMREP) message only if the node has fresh enough route metric for the destination. When generating a RMREP message, a node copies entries (e.g., RM_Orig, mid, IP_Dst, IP_Orig, and Seq_Orig) from the RMREQ message into the corresponding fields in RMREP message. In addition a node generating a RMREP message adds its neighbor list to the RMREP message.

If a node generating a RMREP is the destination itself, the node increments its own sequence number only if the current sequence number is the same as the destination sequence number carried in the RMREQ. Otherwise the destination node does not change the sequence number. The destination node places its own (perhaps newly incremented) sequence number into the destination sequence number field of the RMREP and set the RM_Dst field to zero and broadcasts the RMREP message. However if a node generating a RMREP is not the destination of a RMREQ but instead has a fresh enough route metric for the destination, the node copies its known destination sequence number for the destination into Seq_Dst field in the RMREP message. The node places its route metric for the destination (indicated by the route metric in the route metric table) into RM_Dst field in the RMREP and broadcasts the RMREP message.

Once generated the RMREP is broadcast towards the originator of the RMREQ. The RM_Orig and RM_Dst will be updated with the forwarding node's route metric as the RMREP is forwarded towards the RMREQ originator.

On receiving a RMREP message, a node updates the existing route metric entry for the destination only if the delivered route metric is fresh enough or has a better route metric. If the current node is not the originator of a corresponding RMREQ message, the node can suppress or forward the received RMREP message. A node consults its route metric table entry for the originator node to determine if the RM-REP message is suppressed or forwarded. The node suppresses the RMREP message if the node has not processed a corresponding RMREQ message or the route metric for the originator is worse than the delivered originator route metric (i.e., RM_Orig) in the RMREP. A node forwards the updated RMREP message by broadcasting it, only if the node has a fresh enough or a better route metric for the originator of the corresponding RMREQ. Algorithm 4 shows the processing steps of RMREP messages when a node i receives RMREP messages.

Algorithm 4 Processing RMREP($s, mid, d, D^d, S^d, D^s, S^s$) at node i

```
$s$ is a copy of the originator IP address in RMREQ
$mid$ is a copy of $mid$ in RMREQ
$d$ is a copy of the destination IP address in RMREQ
$D^d$ is RM_Dst and $D^s$ is RM_Orig in RMREP
if $i = d$ then
  // $i$ is the destination of RMREQ
  update $s$'s RM if RMREP carries fresh enough or better route metric for $s$
  drop RMREP
else if $i = s$ then
  // $i$ is the RMREQ originator
  update $d$'s RM if RMREP carries fresh enough or better route metric for $d$
  drop RMREP
else
  // $i$ is an intermediate node
  if $S_i^d < S^d$ then
    // $i$ has an obsolete route metric for $d$
    $fresher\_dest\_rm = 1$
    $S_i^d = S^d$
    $D_i^d = D^d + 1$
  else if $(S_i^d = S^d) \&\& (D_i^d > D^d + 1)$ then
    // better RM for $d$
    $better\_dest\_rm = 1$
    $D_i^d = D^d + 1$
  end if
  if $S_i^s < S^s$ then
    // $i$ has an obsolete metric for $s$
    $S_i^s = S^s$
    $D_i^s = D^s + 1$
    drop RMREP
  else if $S_i^s = S^s$ then
    if $D_i^s < D^s$ then
      if $rep\_state(s, mid) \neq Replied$ then
        broadcast RMREP($s, mid, d, D_i^d, S_i^d, D_i^s, S_i^s$)
        $rep\_state(s, mid) = Replied$
      else
        //already replied one but need to resend it because of fresher or better RM for $d$
        if $fresher\_dest\_rm || better\_dest\_rm$ then
          broadcast RMREP($s, mid, d, D_i^d, S_i^d, D_i^s, S_i^s$)
        end if
      end if
    else
      if $D_i^s > D^s + 1$ then
        $D_i^s = D^s + 1$
      end if
      drop RMREP
    end if
  else
    // $S_i^s > S^s$
    if $rep\_state(s, mid) \neq Replied$ then
      broadcast RMREP($s, mid, d, D_i^d, S_i^d, D_i^s, S_i^s$)
      $rep\_state(s, mid) = Replied$
    else
      already replied one but need to resend it because of fresher or better RM for $d$
      if $fresher\_dest\_rm || better\_dest\_rm$ then
        broadcast RMREP($s, mid, d, D_i^d, S_i^d, D_i^s, S_i^s$)
      end if
    end if
  end if
end if
```

4.4 Data Delivery

Once the route metric for the destination is determined, data packets are delivered through broadcast without following any pre-selected path. BRP's data plane design has four key challenges. First, nodes must agree that nodes closer to the destination have higher chance (e.g., priority) of forwarding data packets to avoid redundant duplication. BRP neither exchanges nor maintains network topology information. Therefore each node must independently decide its forwarding chance in distributed manner without knowing whole network topology. Second, data packets are delivered through broadcast and multiple nodes can overhear and forward the packets. This multiple forwarding can cause many redundant duplications in large dense networks. Thus BRP must provide a scheme such that the closest receiver to the destination forwards data packets and other nodes suppress the packets. Third, BRP must avoid simultaneous transmissions by multiple receivers to minimize collisions. Finally, data packets are broadcast and vulnerable for packet losses and corruptions since wireless MAC layer does not support reliability for broadcast packets. Therefore BRP must efficiently detect and recover lost packets.

BRP operates on batches of data packets in order to reduce the communication cost of agreement of forwarding. The source node adds its route metric for the destination to data packets and broadcasts them. Here BRP *neither* designate next hop *nor* forwarder list as in ExOR. Upon receiving packets, nodes first check if their route metric for the destination is smaller than the route metric carried in packets. If it is, they buffer successfully received packets and await the end of batch and a small amount of backoff time. Otherwise they discard the received packets. Nodes then forward only packets that are not acknowledged in the batch maps of higher priority nodes for the destination: these transmissions are called *fragment* of the batch. Each

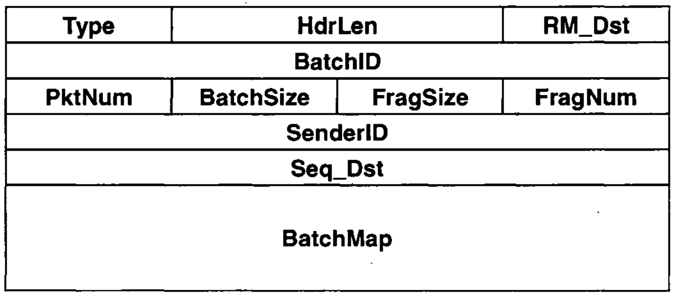


Figure 4.2. BRP Data Packet Header

packet also carries the sender's batch map that indicates the received packets at the sender. Moreover the destination sends destination acknowledgements (DACKs) to the source using batch map that indicates which packets are correctly delivered.

4.4.1 States Maintained at Nodes

Each BRP node is expected to store the following state information of each batch that the node participates in forwarding. The *packet buffer* stores the received packets in the current batch. The *forwarding timer* indicates the time at which the node starts forwarding packets in the packet buffer. This forwarding time is decided based on the route metric for the destination. The *forwarding tracker* records which packets are further progressed towards destination. Further progressed packets mean that nodes closer to the destination forward the packets. These further progressed packets are removed from the packet buffer at the current node.

4.4.2 BRP Data Packet Header

Each data packet carries the BRP data packet header as shown in Figure 4.2. The *Type* field indicates the packet type such as data or destination acknowledgement. The *HdrLen* field indicates the length of BRP data packet header. The *RM_Dst* and *Seq_Dst* carry the route metric and the destination sequence number for the destination, respectively. The *BatchID* and *BatchSize* fields are the current batch identifier and batch size. The *PktNum* is the packet offset in the current batch. The *FragSize* field indicates the size of the currently sending node's fragments (in packets), and the *FragNum* is the current packet's offset within the fragment. The *SenderID* carries the currently sending node's IP address. The *BatchMap* is a copy of the sending node's batch map. This batch map is used for tracking forwarded packets (e.g., aggregated passive acknowledgement) and each entry is binary value (e.g., zero or one). One means that the corresponding packets are further progressed by the sender or higher priority nodes.

4.4.3 Batch Preparation

The source collects a batch of packets for the same destination from upper layer and chooses a unique batch ID for them. Therefore each batch is uniquely identified with the combination of the source and the batch ID. The source then adds BRP data packet header to each packet and fills out all fields in the BRP data packet header. Here Type field is set to data. The source also copies the destination route metric and the destination sequence number from its route metric table to RM_Dst and Seq_Dst entries. If the destination route metric is not available, the source originates a route metric request message. The source adds a batch map whose all fields are set to one. The source updates the packet number and the fragment fields as packets in the batch are transmitted. Finally the source broadcasts packets in the batch.

4.4.4 Processing Data Packets

Upon receiving data packets, a node first compares the destination sequence number and the route metric carried in the packets to the stored destination sequence number and the stored route metric at its route metric table. If the stored route metric is obsolete (e.g., the stored destination sequence number is smaller than the carried destination sequence number), the node discards the packet. Also, duplicated packets are discarded. However, new packets are buffered in packet buffer and the node updates the corresponding batch map. In addition if the received packets are forwarded by higher priority nodes [§], the node merges the carried batch map to the stored batch map and removes the buffered packet. In this way the lower priority node avoids forwarding the packets that are already forwarded by higher priority nodes.

Whenever the destination completes receiving fragments from neighbors or the passive acknowledgement (PACK) time out occurs, the destination sends three PACK packets that contain only the destination's batch map. The destination's batch map indicates which packets are correctly received. When these PACK packets arrive at the fragment forwarding nodes, the nodes retransmits undelivered fragments. In addition the destination sends k destination acknowledgements (DACKs) to the source when the destination receives all packets in a batch or no packets are delivered during predefined waiting time. We use ten for k in our simulations. Upon receiving DACKs, the source sends undelivered data packets and new packets in a new batch.

4.4.5 Scheduling

BRP schedules the time at which nodes start forwarding their fragments on the basis of batch. This schedule attempts to give higher priority to the nodes closer to the destination. When a node receives packets of a batch, the node estimates the fragment completion time based on the packet inter-arrival time and waits a small amount of backoff time before start forwarding. The backoff time is calculated based on the route metric to the destination at the node. First, nodes estimate the number of potential receivers among the receiver's neighbors. These potential receivers are common neighbors of the sender s and the current node i (e.g., $N_i \cap N_s$). With high

[§]The carried route metric in the packet is smaller than the stored route metric for the destination.

probability (we assume that nodes are uniformly distributed in the network), half of the potential receivers have the same route metric as the sender's route metric and half of them have the same as the receiver's route metric for the destination. Therefore only half of the potential receivers need to forward the received packets. This means that half of the potential receivers compete for transmissions. Therefore we consider assigning backoff time to avoid collision among $|N_i \cap N_s|/2$ nodes at each node i. Second, we assign the maximum backoff time (ξ) of $\delta \times |N_i \cap N_s|/2$. Here δ is a predefined small amount of time and we use $10ms$ for the simulations. Third, we split ξ into two parts: one is ξ/k and the other is $(\xi - \xi/k)$. ξ/k fraction is used for further progressed nodes whose route metrics are smaller than other potential receivers. For example, the route metrics at receivers are smaller than the carried route metric by more than one. Finally, we assign uniformly distributed backoff time within the fraction of the maximum backoff time for each node: $U(\xi/k)$ and $U(\xi - \xi/k))$ for further progressed receivers and other receivers, respectively. We use 5 for k in our simulations.

4.5 Greedy Set Selection (GSS) Algorithm

In route metric discovery process for a destination, RMREQ messages should be delivered to the destination or at least one node that has fresh enough route metric (RM) for the destination. At the early stage of network bootstrap, RMREQ originators do not have any knowledge of network topology and RMREQ messages theoretically must be delivered to all nodes in the network. That means bubbles should cover all nodes in the network. Here a bubble represents a communication range at each node. The simplest way to deliver RMREQ messages to all nodes in the network is controlled flooding, but controlled flooding causes redundant rebroadcasts. In addition to "worst-case" bound provided by controlled flooding, a theoretical "best-case" bound is provided by the minimum connected dominating set (MCDS). The MCDS presents the smallest set of rebroadcasting nodes such that the set of nodes is connected and all non-set nodes are within one-hop of at least one member of the MCDS. Determination of an MCDS is an NP-hard problem [32] and many approximation algorithms have been proposed. [53, 58, 62, 77]

The key problem in delivery (e.g., covering) of broadcast messages is how to choose appropriate nodes that minimize redundant rebroadcasts. This is the same as how to independently assign backoff time to each node without knowing the network topology in a distributed manner when a node receives a RMREQ message. Nodes chosen for rebroadcast must have less backoff time and forward the message earlier than other nodes. Also when neighbor nodes receive that broadcast message, they suppress the previously received broadcast message. This problem is different from the coverage problem because of the following two constraints: 1) the originator of broadcast (e.g., RMREQ) message is fixed and 2) bubbles should intersect with at least one bubble that is already chosen. Bubble intersection means that two nodes should be in the communication range of each other. Many papers have tried to address this issue but they assume centralized control to select the next rebroadcasting node and do not provide the way how each node independently decides if it should rebroadcast the received broadcast message or not in distributed manner with no knowledge of the network topology. [53, 58, 62, 77, 44]

In this section we propose and evaluate a novel greedy set selection (GSS) algorithm that automatically selects the best candidates in order to minimize unnecessary rebroadcast messages. Selection of the best candidates is transformed to how to efficiently assign backoff time in distributed manner without knowledge of the network topology at each node. GSS utilizes one-hop neighbor information and assigns backoff time based on neighbor set differences.

The basic goal of GSS is to select the smallest number of bubbles (i.e., nodes) to deliver a RMREQ message to all nodes in the network. Before we describe GSS, we first devise a method to estimate the lower and upper bound of the number of bubbles to deliver a RMREQ message to all nodes in a network. As described earlier to find a MCDS is a NP-hard problem. Therefore we devise a simple method to identify the lower and upper bound of the number of bubbles for a given network area by assuming square area communication range.

4.5.1 Square Area Communication Range

To estimate the lower and upper bound of the number of bubbles to deliver a RMREQ message to all nodes in a network, we develop a simple analytical model. Here we make several assumption. First we assume that the network area is rectangular $(A = X \times Y)$. Second assumption is that a node's communication range is a square and we consider two squares (e.g., inner and outer squares) as seen in Fig 4.3. We use an inner square (e.g., S_i) to estimate upper bound of the number of bubbles and an

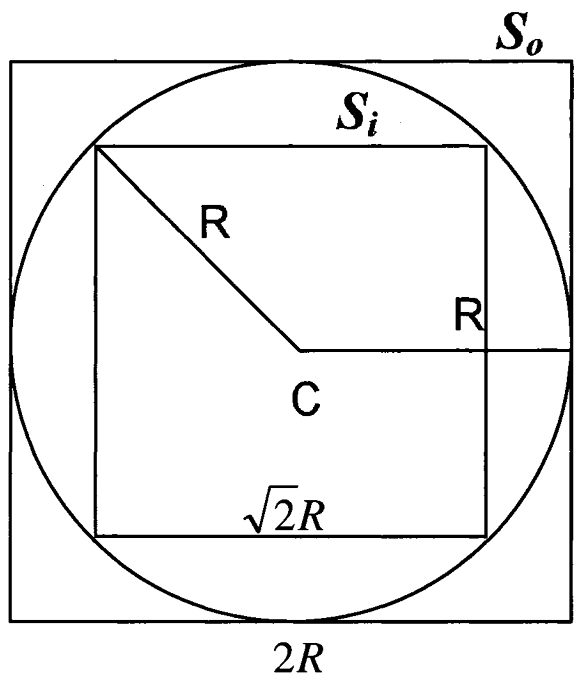


Figure 4.3. Communication Range of a Node

outer square (e.g., S_o) to estimate lower bound of the number of bubbles to deliver a RMREQ message to all nodes in a network. The length of the outer square (S_o) is $2R$ and the length of the inner square is $\sqrt{2}R$ compared to the circle of the radius R as shown in Fig 4.3.

The number of bubbles that are used for delivering a RMREQ to all nodes in a network purely depends on the network area and the length of the squares. We estimate the minimum number of bubbles to cover whole network area using S_i and S_o squares. The number of bubbles with S_i communication range will be the upper bound of the number of bubbles and the number of bubbles with S_o communication range will be the lower bound.

The upper bound of the number of bubbles for the network area ($A = X \times Y$)

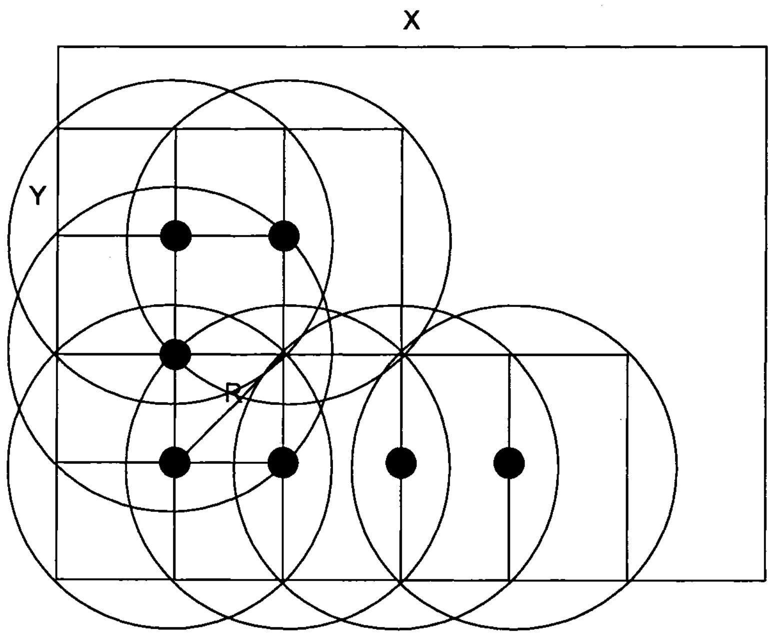


Figure 4.4. Network Area Coverage

with S_i communication range is represented as follows.

$$\min(\left\lceil\frac{\sqrt{2}X}{R}\right\rceil \times \left\lceil\frac{\sqrt{2}Y}{2R}\right\rceil - 1, \left\lceil\frac{\sqrt{2}X}{2R}\right\rceil \times \left\lceil\frac{\sqrt{2}Y}{R}\right\rceil - 1) \tag{4.1}$$

In addition the lower bound of the number of bubbles for the same network area with S_o communication range is defined as follows.

$$\min(\left\lceil\frac{X}{R}\right\rceil \times \left\lceil\frac{Y}{2R}\right\rceil - 1, \left\lceil\frac{X}{2R}\right\rceil \times \left\lceil\frac{Y}{2}\right\rceil - 1) \tag{4.2}$$

4.5.2 Greedy Set Selection (GSS) Algorithm

In this section we describe the proposed greedy set selection (GSS) algorithm. The key idea of GSS is to choose a node whose number of uncovered neighbors are largest. That means GSS chooses a node with the largest number of neighbors that have not received a RMREQ message. Likewise GSS maximizes the node (*not area*) coverage since a newly chosen node covers the largest number of new nodes at each step.

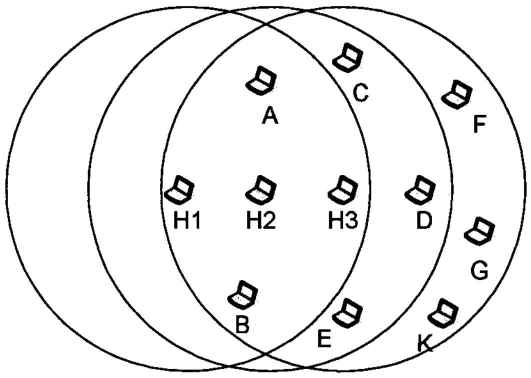
Table 4.2. Notations in GSS

Notation	Definition
Ω	A set of whole nodes in the network
$\mathcal{ONS}$	A optimal node set
$\mathcal{C}$	A set of nodes covered by nodes in $\mathcal{ONS}$
$\mathcal{NC}$	A set of next candidate nodes($\mathcal{C} \setminus \mathcal{ONS}$)
$\mathcal{UC}_i$	A set of node i's uncovered neighbors ($\mathcal{UC}_i = N_i \setminus \mathcal{C}$)

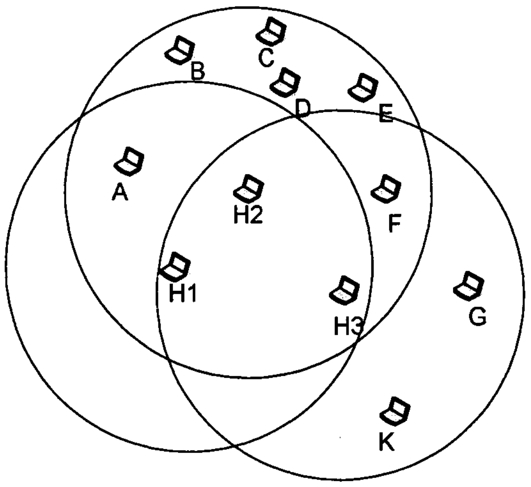
Figure 4.5 shows the example of node selection in GSS algorithm. We assume that node H1 broadcasts a control message (e.g., RMREQ), and H2 and H3 have received the message. In Figure 4.5(a), the number of H2's uncovered neighbors are three (nodes C, D, and E) and the number of H3's uncovered neighbors are six (nodes C, D, E, F, G, and K). Therefore GSS algorithm chooses H3 as a next forwarder since H3 can cover three more neighbors than H2. Note that H2 suppresses the received broadcast messages since its all neighbors are covered by H1 and H3 when it receives the forwarded message from H3. Figure 4.5(b) shows that GSS focuses on the number of uncovered neighbors instead of area. Here H2 can cover less additional area than H3. However the number of H2's uncovered neighbors are larger than the number of H3's uncovered neighbors. H2 has five uncovered neighbors (e.g., node B, C, D, E, and F) while H3 has three uncovered neighbors (e.g., node F, G, and K). Therefore GSS algorithm first chooses H2 for the next forwarder in this case.

We summarize the notation used in GSS algorithm in Table 4.2 and describe GSS algorithm on step by step as follows.

```
1 choose a source node $s$ in a network
2 add $s$ to $\mathcal{ONS}$: $\mathcal{ONS} = \mathcal{ONS} \cup \{s\}$
3 add $s$ and $N_s$ to $\mathcal{C}$: $\mathcal{C} = \mathcal{C} \cup \{s\} \cup N_s$
4 foreach $i \in N_s$
```



(a)



(b)

Figure 4.5. Node Selection in GSS Algorithm

```
  add $i$ to $\mathcal{NC}$ only if ($i \notin \mathcal{ONS}$ AND $i \notin \mathcal{NC}$): $\mathcal{NC} = \mathcal{NC} \cup \{i\}$
5 update $\mathcal{UC}_i$ for $i \in \mathcal{NC}$: $\mathcal{UC}_i = N_i \setminus \mathcal{C}$
6 choose a node $j$ such that $MAX_{arg}|\mathcal{UC}_k|$ where $k \in \mathcal{NC}$
7 $s = j$
8 $\mathcal{NC} = \mathcal{NC} - \{j\}$
9 repeat step 2 through step 8 until $\mathcal{C} = \Omega$
```

We measure the performance of GSS algorithm and compare it to ones with controlled flooding and square communication ranges. Three assumptions are made to expedite analysis without loss of the generality. First, we assume that the network is not partitioned. If the network is partitioned a broadcast (e.g., RMREQ) message can not be delivered to all nodes in the network. Second, we assume that the whole network topology is already known. This means that the neighbor set of all nodes in the network is known. This assumption is valid if the centralized control is assumed as in other proposed algorithms. However we extend GSS algorithm to distributed manner in implementation since each node must independently decide waiting (back-off) time before it forwards the RMREQ message without knowledge of the network topology. Third assumption is that the communication range of each node is circular (e.g., $radius = R$). This assumption is different from square communication range but is mainly used in wireless networks.

We use uniformly distributed random networks for the measurements. We run 50-time simulations for each topology and measure the average value for each metric. We run simulations with various network area and various network density and we present the representative simulation results in the network area of $1500 \times 1500m^2$

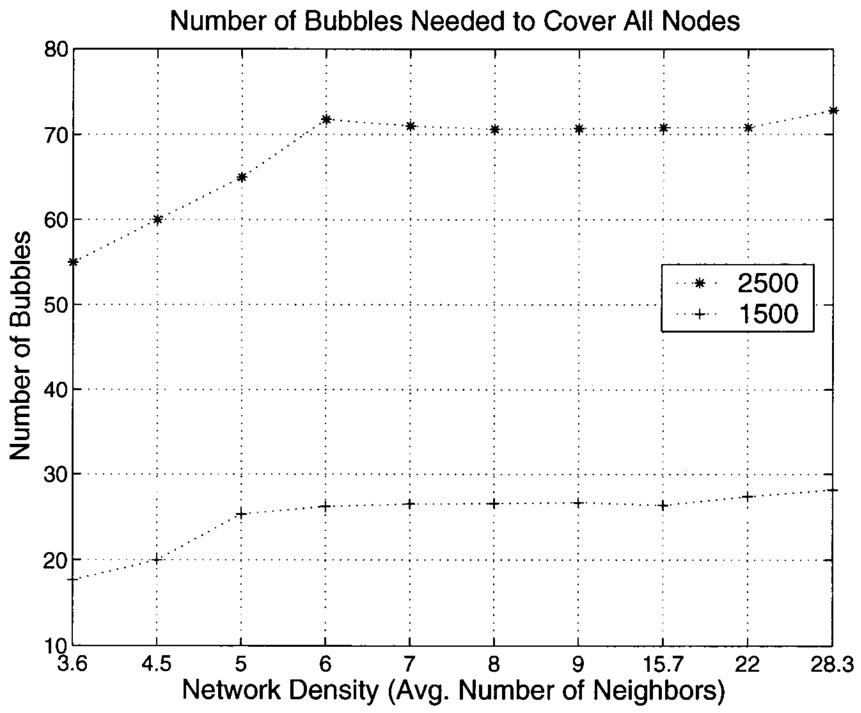


Figure 4.6. The Number of Bubbles Required with GSS

and $2500 \times 2500m^2$. We uniformly distribute nodes on the network area with various network density (e.g., 3.6, 4.5, 5.0, 6.0, 7.0, 8.0, 9.0, 15.7, 22.0, and 28.3). Here we calculate the network density ρ (e.g., the average number of neighbors) as follows: $\rho = N \times \frac{\pi \times R^2}{X \times Y}$, where N is the number of nodes in a network, R is the communication range, X is the length of the network area, and Y is the height of the network area.

First we measure the number of bubbles required to cover whole nodes in the network. Coverage means that a broadcast message (e.g., RMREQ) is delivered to all nodes in the network. The number of bubbles with S_i (inner square) communication range is 44 and 119 for $1500 \times 1500m^2$ and $2500 \times 2500m^2$ network areas, respectively, which are the upper bound of the number of bubbles. The number of bubbles with S_o (outer square) communication range is 17 and 49 for two network areas, which are the lower bound of the number of bubbles.

Figure 4.6 shows that the number of bubbles (i.e., nodes) used to cover all nodes

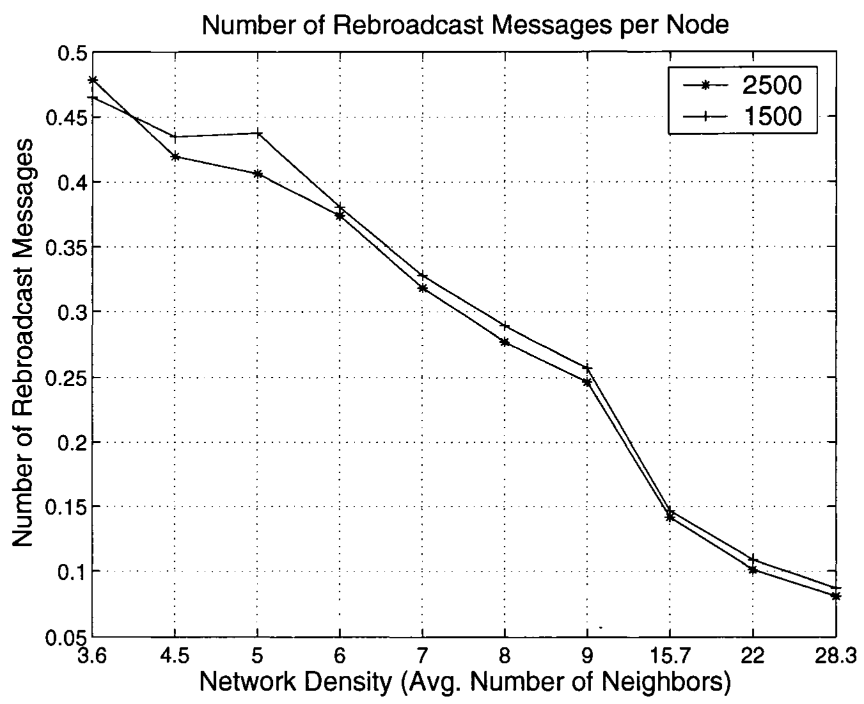


Figure 4.7. The Number of Rebroadcast Messages per Node

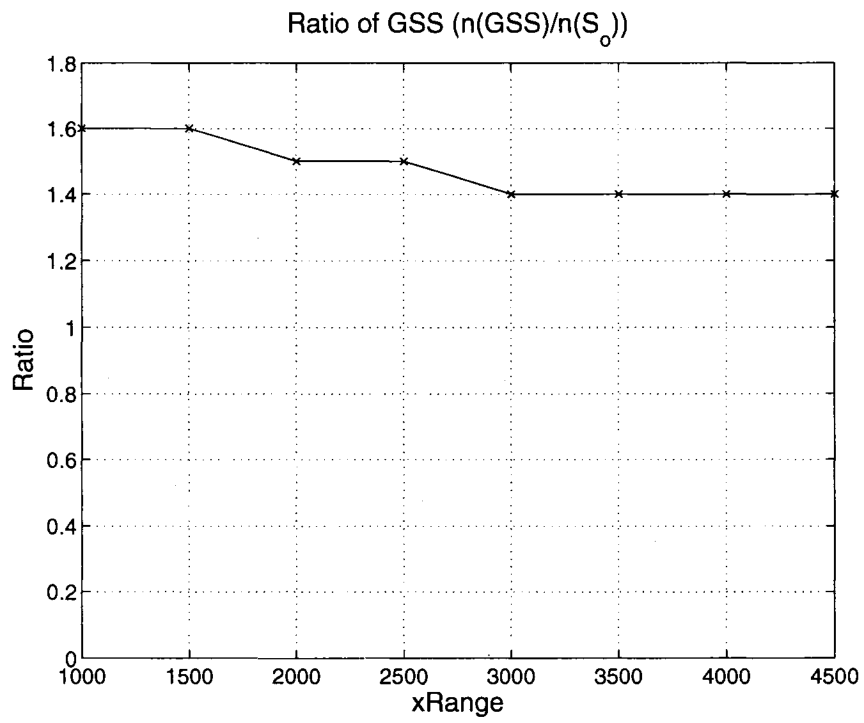


Figure 4.8. Ratio of the Number of Bubbles with GSS to S_o

in the network converges and does not change much after the network density of five. When the network density is bigger than 5, each node covers more nodes as network density increases and the total number of nodes that covers whole network nodes does not change much. Second, we measure the number of rebroadcast messages per node. As seen in Figure 4.7, each node rebroadcasts less than 0.5 messages with GSS algorithm while one message with controlled flooding (e.g., AODV). We can reduce the number of redundant rebroadcast messages (e.g., RMREQ) more than 50% under the network density of 3.6 and the reduction significantly increases up to more than 90% with the increase of the network density.

Finally, we estimate the ratio of the number of bubbles required in GSS to the theoretical lower bound as estimated in expression 4.2. We change the network area and the number of nodes but fix the network density since the network density does not have much effect on the number of nodes covering whole network nodes when the density is greater than 5 as shown in Figure 4.6. We choose the network density of 9 in the measurements. We also estimate the ratio of the number of bubbles (e.g., nodes) with GSS algorithm to the number of bubbles with S_o communication range (e.g., $n(GSS)/n(S_o)$). Figure 4.8 shows that the number of bubbles used with GSS algorithm is about 1.5 times as many as bubbles with S_o communication range that is the theoretical lower bound.

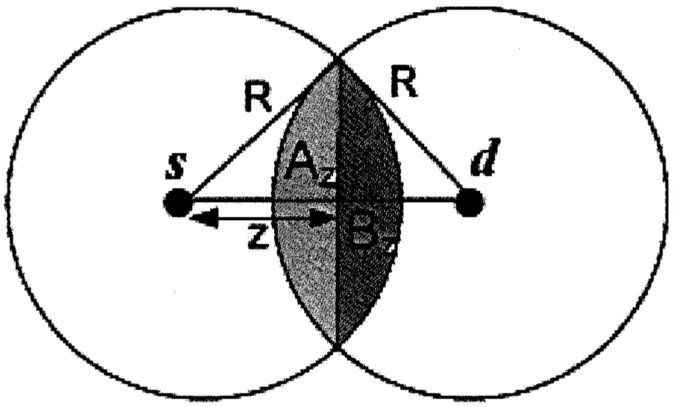


Figure 4.9. Illustration of A_z and B_z

4.6 Comparative Study Based on Analytical Models

In this section we employ a common theoretical framework to derive theoretical performance for BRP and traditional routing protocols (e.g., AODV). We quantify the probability of routing/forwarding failure. The following analysis is devoted to the derivation of the expected number of next potential forwarders and the node/link failure rate. We first estimate the expected number of neighbor nodes that can be next potential forwarders for a destination from a source node that are $2 - hop$ apart. Based on the expected number of neighbor nodes we calculate the probability of routing/forwarding failure. Furthermore we measure the probability of routing/forwarding failure on source-destination pairs that are $k - hop$ apart.

We assume that N nodes are in a multihop wireless ad hoc network and the ad hoc network is in a rectangular region of dimension $l \times h$ (e.g., $1500 \times 300m^2$). In addition we assume that each node communicates with another mobile node with a maximum communication range R. To estimate the number of potential forwarders we first estimate what is the area where next potential forwarders can locate for a destination.

For simple exposition without losing generality we assume that s is a sender and

d is a destination as in Figure 4.9. d can be reached through $2-hop$ distance vector from s. We estimate the average area where s's next potential forwarders can locate for d. The overlapped region between the communication range of s and d is the area where s's next potential forwarders can locate and the area can be measured as $A_z + B_z$ in Figure 4.9. We can measure A_z and B_z as in Equation 4.3. Here $A_z = B_z$ since we assume that communication range of s and d is the same.

$$A_z = B_z = R^2 \left[\cos^{-1}(z/R) - (z/R)\sqrt{1-(z/R)^2}\right] \tag{4.3}$$

We estimate an expected area of $A_z + B_z$ by integrating $A_z + B_z$ from 0 to R as in Equation 4.4.

$$\bar{A} = \frac{1}{R}\int_0^R (A_z + B_z)dz \tag{4.4}$$

In addition we estimate the number of potential forwarders ($\bar{n_1}$) for a destination as in Equation 4.5 where A is the region (e.g., $l \times h$) of the ad hoc network in the analysis. Here we assume that N nodes are uniformly distributed in the network area of A.

$$\bar{n_1} = N \times \frac{\bar{A}}{A} \tag{4.5}$$

We define $Pr_{fail}^{BRP}(k)$ and $Pr_{fail}^{trad}(k)$ as the probability of routing/forwarding failure when s is $k-hop$ apart from d with BRP and traditional routing protocols respectively. We can estimate the probability of routing/forwarding failure of $Pr_{fail}^{BRP}(2)$ and $Pr_{fail}^{trad}(2)$ as in Eq. 4.6 and Eq. 4.7, respectively. p represents a node/link failure rate in our analytical model and intuitively $Pr_{fail}^{trad}(1) = Pr_{fail}^{BRP}(1) = p$ where both s and d are within each other's communication range.

$$Pr_{fail}^{BRP}(2) = 1 - (1 - p^{\bar{n_1}})(1 - p) \tag{4.6}$$

$$Pr_{fail}^{trad}(2) = 1 - (1 - p)^2 \tag{4.7}$$

Likewise we can derive the probability of routing/forwarding failure when s is *k-hop* away from d as Eq. 4.8 and Eq. 4.9 for BRP and traditional routing protocols, respectively. Here $\bar{n_i}$ is the expected number of neighbors that are $i - hop$ and $(k - i) - hop$ apart from s and d, respectively.

$$\begin{aligned} &Pr_{fail}^{BRP}(k) = 1 - (1 - p)\prod_{i=1}^{k-1}(1 - p^{\bar{n_i}}), \\ &where\, \bar{n_i} = f(N, A, i) \end{aligned} \tag{4.8}$$

$$Pr_{fail}^{trad}(k) = 1 - (1 - p)^k \tag{4.9}$$

We measure the probability of routing/forwarding failure with the network area of $1500 \times 300m^2$, the communication range of $250m$, and the node/link failure rate of 0.05 and 0.25. The number of nodes in the network varies such as 10, 28, and 50, and the average network density (ρ) is 4, 12, and 21, respectively. The average network density means the average number of neighbors for each node. Figure 4.10 shows the probability of routing/forwarding failure with varying network density and with the node/link failure rate of 0.25. When the network density is four, the rout-

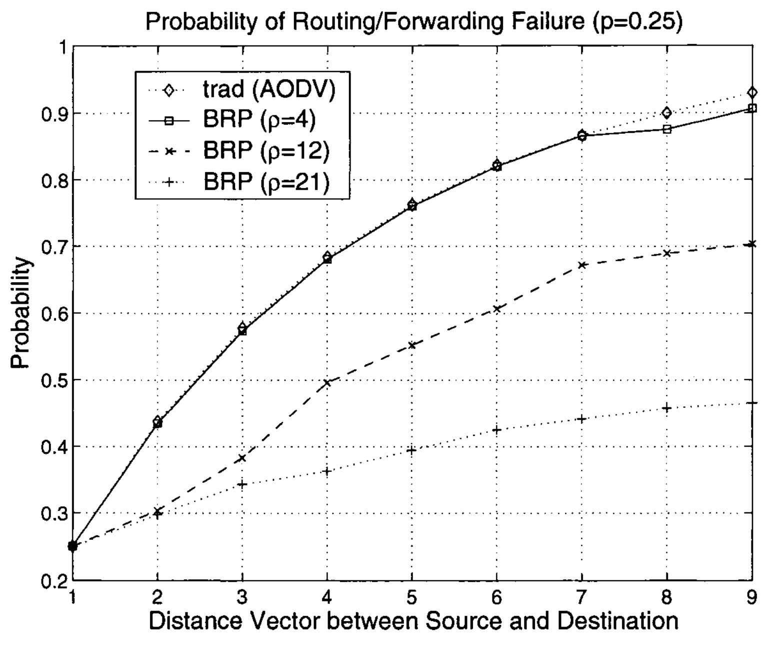


Figure 4.10. Probability of Routing/Forwarding Failure on Various Network Density

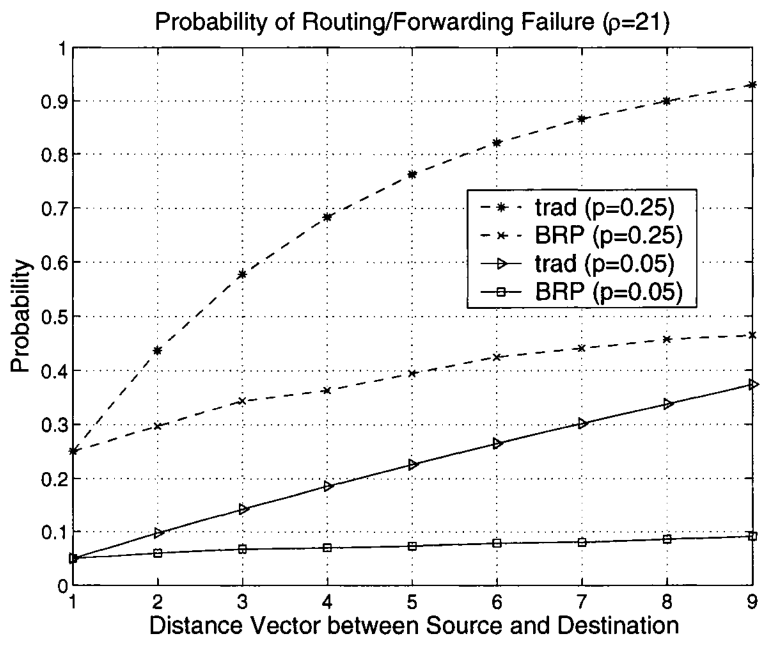


Figure 4.11. Probability of Routing/Forwarding Failure on Various Link Failure Rate

ing/forwarding failure of BRP is almost the same as the traditional routing protocol. When the network density is four, the expected number of neighbors at each node for a destination is generally one. That is the reason why the two probabilities are almost the same. However, when the network density increases to twenty one, the probability of routing/forwarding failure decreases more than 40% with BRP while the probability of the traditional routing protocol stays.

Figure 4.11 shows the probability of routing/forwarding failure with the different node/link failure rate (e.g., 0.05 and 0.25). Here we fix the number of nodes to 50 where the network density is 21. BRP decreases the probability of routing/forwarding failure that converges to 0.4 and 0.1, and reduces the failure probability up to more than 40% and 30% with the node/link failure probability of 0.25 and 0.05, respectively.

4.7 Performance Measurement

In this section we present performance evaluation results from extensive simulations in ns2 [54]. We compare the performance of the control plane (greedy set selection algorithm) design to controlled flooding (e.g., AODV) mainly used in most wireless routing protocol, scalable broadcast algorithm (SBA) [58], and gossip routing [33]. In gossip routing, each node only rebroadcasts with a predefined forwarding probability. On SBA, each node waits a randomly assigned delay before rebroadcasts based on the ratio of the maximum degree among its neighbors ($d_{N_{max}}$) to the number of neighbors (d_{me}) (i.e., $\frac{d_{N_{max}}}{d_{me}}$). Since gossip routing was proposed for control plane [33], the performance comparison of data plane is executed with BRP, AODV and SBA excluding gossip routing. We choose 0.75 for the forwarding probability (γ) in gossip routing since 0.72 ensures that almost all nodes get the message in grid networks of network density four. [33] We exclude the performance comparison of BRP and ExOR [13] since ExOR addressed data plane issue and utilized existing link state routing protocol for control plane while BRP mainly focuses on control plane with opportunistic forwarding. We use regular networks and wireless mesh network (e.g., Roofnet [67]) in the simulations.

We set data plane parameters defined in section 4.4.5 for all simulations as follows: the batch size of 100, the maximum backoff time parameter δ of $10ms$, and the backoff time split factor k of 5. We use 250 meter for the communication range of each node according to default physical parameters in ns2. The radio propagation model is the two-way ground model [64]. We study the performance of following metrics, of which the first two were also used in [15].

- The *number of rebroadcast RMREQ messages* represents the average number of rebroadcast RMREQ messages at each node.

- The *end-to-end packet delivery ratio* represents the ratio of the data packets delivered to the destination to those generated by CBR traffic sources.

- The *normalized control message overhead* denotes the number of control (e.g., routing) messages transmitted per data packet delivered at the destination. Control messages include route metric requests, route metric replies, and route error messages.

4.7.1 Regular Networks

In this section we evaluate the efficacy of greedy set selection (GSS) algorithm with static regular networks. We conduct simulations with various network density such as 4, 8, 12, 20, 24, 28, 36, and 44. Nodes are uniformly distributed over a square network area (e.g., $1000 \times 1000m^2$). The number of nodes are assigned on the basis of network density (ρ) (e.g., $N = \rho \times \frac{1000^2}{\pi \times 250^2}$). We assume that no link/node failure occurs in this section and estimate link error effect with real Roofnet topology in section 4.7.2. We choose our traffic sources to be constant bit rate (CBR) sources. A source sends four 512-byte packets per second to a destination. The source and the destination locate at opposite corners of the square network area and they have the same number of neighbors as the network density. We measure the average metrics from twenty 90-second simulations for each network density. First we measure the end-to-end packet delivery ratio. End-to-end packet delivery ratio is more than 0.95 for all routing protocols and no significant difference is among protocols. This mainly attributes to the assumption of no link/node failure.

Second, we measure the average number of rebroadcast RMREQ messages at each node for each RMREQ message and the normalized control message overhead. In this way we measure how many redundant messages can be suppressed at each node. For the suppressed redundant RMREQ messages, we precisely measure the reason of suppression such as all neighbors receive the RMREQ message or a corresponding RMREP message is delivered during the waiting (backoff) period before the rebroadcast. Figure 4.12 and 4.13 show the average number of rebroadcast RMREQ messages at each node and the number of suppressed RMREQ messages respectively. As seen in Figure 4.12, BRP saves redundant rebroadcast RMREQ messages about 20% through 90% compared to controlled flooding (e.g., AODV). With the gossip routing protocol, each node saves the rebroadcast RMREQ messages around $1 - \gamma$ where γ is the predefined forwarding probability (e.g., 0.75) at each node. SBA saves rebroadcasts up to 50%. BRP is adaptive to the network density and shows the best performance of saving redundant rebroadcasts. Gossip routing protocol suppresses RMREQ messages only based on predefined forwarding probability and no suppression occurs in AODV. We therefore measure the reason of suppression of rebroadcast RMREQ messages only for BRP and SBA. BRP shows two through ten times enhancement in suppression of redundant messages than SBA when the network density is greater than 4 as shown in Figure 4.13. BRP mainly suppressed RMREQ messages because all neighbor nodes receive the RMREQ messages (e.g., covered neighbors).

Figure 4.14 shows the control message overhead (i.e., the ratio of the number of control messages to the number of delivered packets at the destination). Interestingly SBA does not look getting benefit much in the control message overhead even though SBA saves up to 50% of RMREQ messages compared to AODV. It mainly attributes to RMREP messages with further inspection. SBA broadcasts RMREP messages and the backoff scheme based on the maximum degree of neighbors and its own degree does not efficiently suppresses the RMREP messages. BRP effectively suppresses redundant control messages and reduces control message overhead more than 60%.

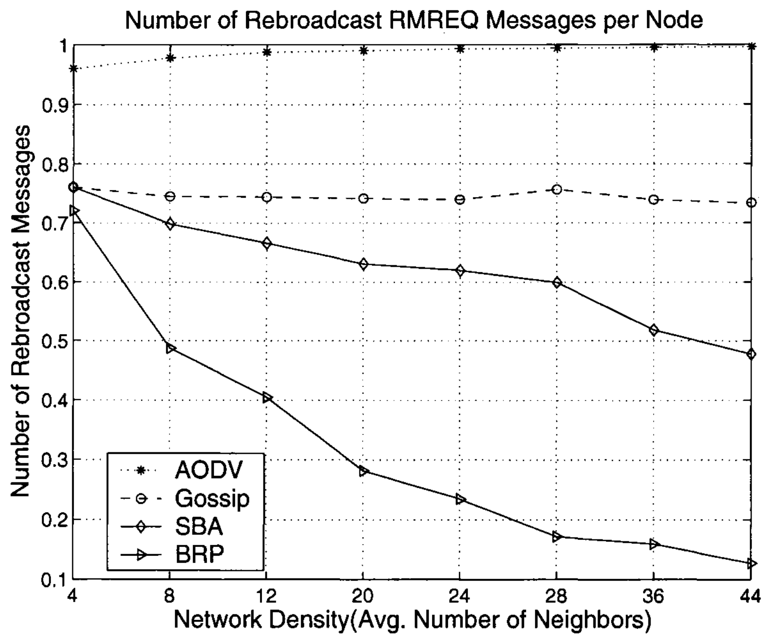


Figure 4.12. Number of Rebroadcast RMREQ Messages per Node on Regular Networks

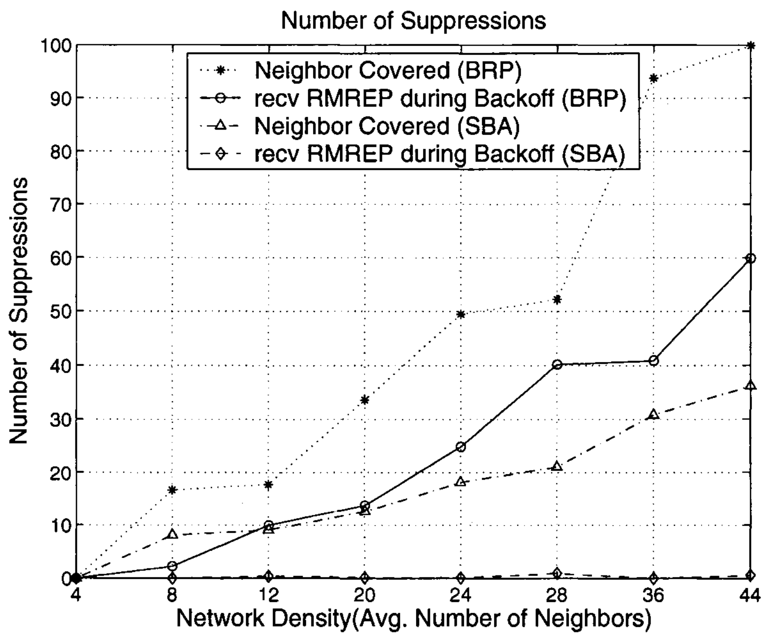


Figure 4.13. Number of Suppressed RMREQ Messages on Regular Networks

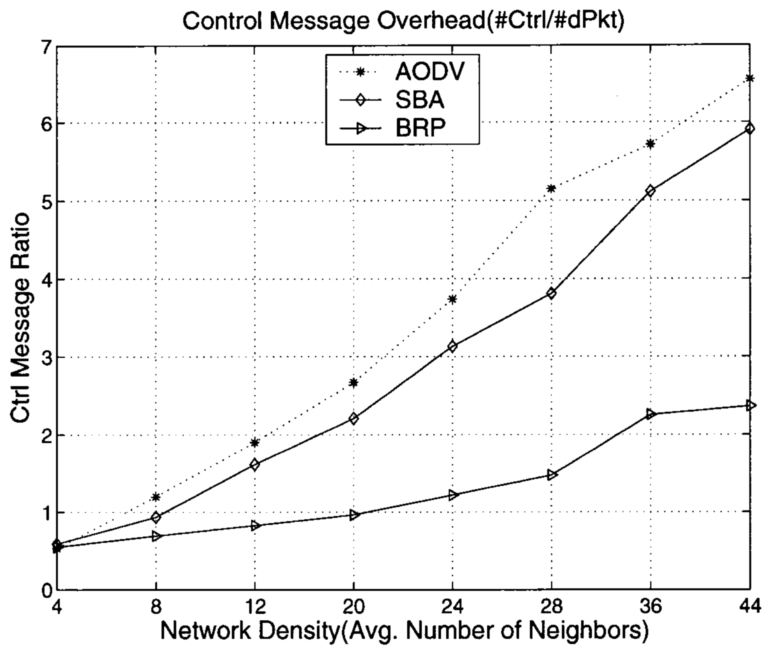


Figure 4.14. Control Message Overhead on Regular Networks

4.7.2 Wireless Mesh Networks

We measure the performance of the control message overhead and data delivery ratio for BRP, AODV, and SBA protocols on real Roofnet topology of thirty eight nodes. We use trace files to retrieve link quality (i.e., link error rate) among pairs of nodes posted in Roofnet [67]. We randomly choose 50 pairs of nodes and measure the performance. In this simulation source nodes send four 1024-byte data packets every second. We run twenty 90-second simulations and measure the average metrics for each source and destination pair.

As seen in Figure 4.15, BRP can save 10% to 60% of the redundant RMREQ messages on Roofnet topology. As the distance vector (i.e., hop count) increases, the number of rebroadcast messages per node also increase since the number of intermediate nodes locating in between the source and the destination, and participating in the route metric discovery increases. Those intermediate nodes rebroadcast the

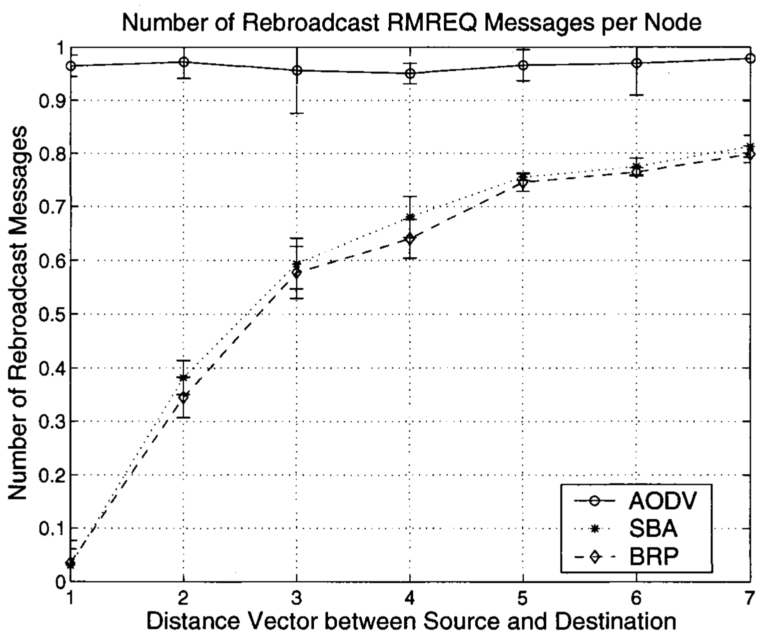


Figure 4.15. Number of Rebroadcast RMREQ Messages per Node on Roofnet

RMREQ messages towards the destination.

BRP resiliently delivers 8% to 30% more packets than AODV as the distance vector between source and destination increases as shown in Figure 4.16. Its gain is mainly due to the reduction of control messages and exploitation of rich spatial diversity on failure-prone links. As seen in Figure 4.17, the performance gap of the control message overhead(i.e., the ratio of the number of control messages to the number of delivered data packets) develops as the source and destination pairs become distant, in which the control message overhead with BRP is less than 1/3 of one with AODV. Also the control message overhead with BRP is up to 40% less than one with SBA.

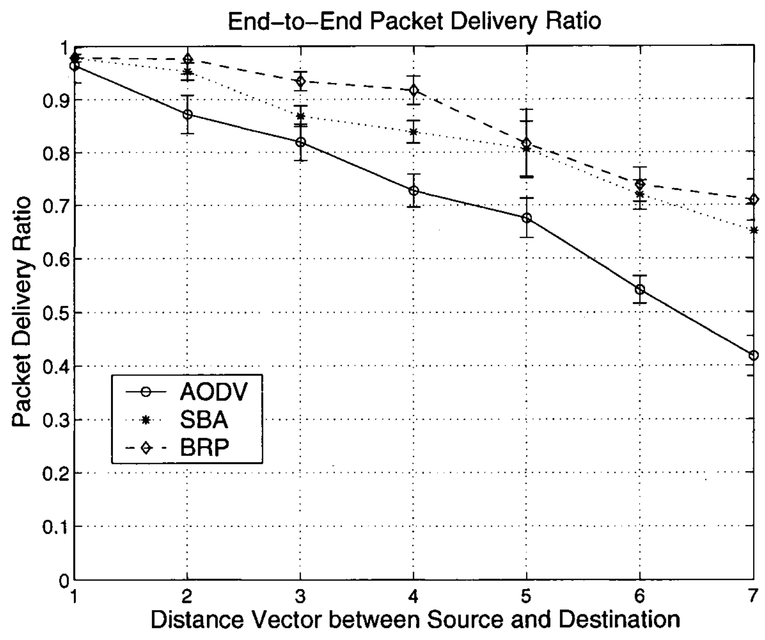


Figure 4.16. End-to-End Packet Delivery Ratio on Roofnet

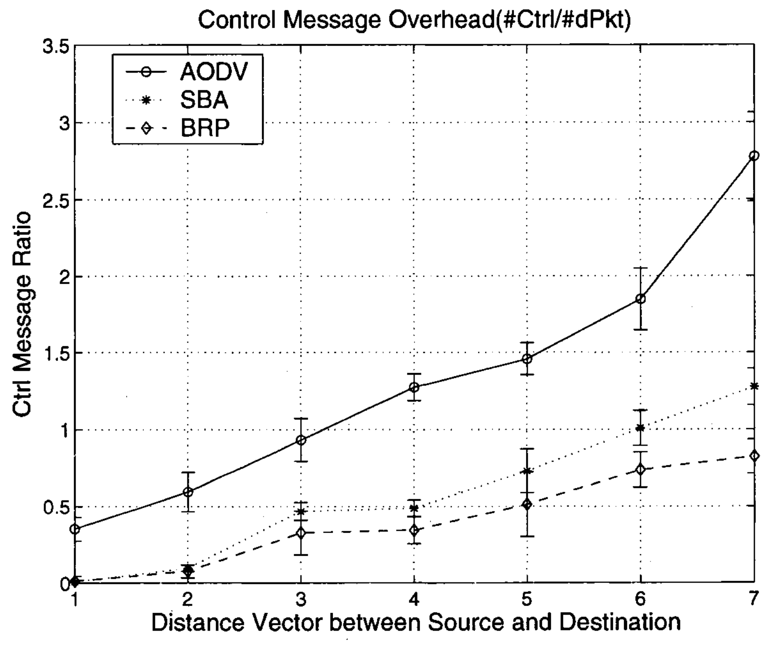


Figure 4.17. Control Message Overhead on Roofnet

4.8 Summary

In this chapter we develop a scalable and opportunistic bubble routing protocol (BRP) that neither utilizes preselected paths nor uses of traditional graph theory to preselect paths. BRP effectively realizes opportunistic forwarding for control messages with greedy set selection (GSS) algorithm and for data packets without knowledge of the network topology. BRP significantly reduces the number of control messages and results in the reduction of control message overhead up to 67% compared to the traditional routing protocols (e.g., AODV). In addition BRP enhances data delivery ratio on the unstable error prone links compared to traditional routing protocol and SBA. Our extensive analysis and simulations demonstrate the effectiveness of BRP in the multihop wireless networks.

Chapter 5

Conclusion

In this dissertation we have addressed the enhancement of key issues on control plane over wired and wireless networks: security in wired networks and resiliency in wireless networks. In particular we focus on systems that prevent and proactively defend against cyber attacks, for example, DoS attacks, and IP spoofing and associated attacks. We also have improved resiliency in multi-hop wireless ad hoc networks through a scalable and opportunistic routing protocol.

In first part, we have developed two security schemes, secure name service (SNS) and lightweight Internet permit system (LIPS), to protect the critical Internet services and resources (e.g., web server, database server or web storage) from unauthorized access and service abuse. SNS preserves the critical Internet services and resources from unauthorized access and DoS attacks through *resource virtualization* and *authenticated packet forwarding*. Resource virtualization is implemented through a secure name service that implements dynamic name binding. This dynamic name binding helps to easily adapt to ongoing cyber attacks at the service level since SNS can revoke a previously assigned virtual ID. Authenticated packet forwarding controls the access to the critical resources and only packets carrying a legitimate security authenticator are allowed to access resources. In addition SNS efficiently distributes the packet authentication overhead to the nodes (e.g., security checkpoints and security gateways) on the path of a source-destination pair. We also address the performance and scalability issues in the authenticated packet forwarding. Our prototype implementation of the authenticated packet forwarding demonstrates the feasibility of implementing SNS on regular Linux system.

LIPS protects Internet services and resources from unwanted traffic by enforcing traffic origin accountability through *access permits*. When a source wants to communicate with a destination, the source must obtain an access permit from the destination before it originates any packets and *identify* themselves to the destination during this access permit request process. This access permit request process efficiently stops randomly spoofed IP packets (e.g., random scanning or probing packets) since the randomly spoofed packets can not carry a valid permit. In addition LIPS facilitates and simplifies the detection of unauthorized intrusion and attacks by enforcing malicious hosts to first request access permits and identify themselves to the intended target hosts before launching attacks. Furthermore by monitoring permit request traffic, LIPS easily locates attack sources or compromised zombie machines in local domain. Unlike other existing PKI systems or shared key systems, LIPS never shares any credential among nodes. Instead only destinations locally maintain their own credentials (e.g., secret keys) and utilize them to issue or verify access permits. Therefore LIPS is much more scalable. Our implementation and experiments have demonstrated the effectiveness of LIPS on regular Linux system.

In part two, we have developed a scalable and opportunistic bubble routing protocol (BRP) that enhances resiliency by exploiting rich (path) spatial diversity over multihop wireless networks. Unlike most existing wireless routing protocols, BRP never utilizes preselected paths but allows neighbors with better chance to forward packets. BRP delivers packets through broadcast and implicitly utilizes possible local multiple paths simultaneously. This simultaneous usage of implicit multiple paths circumvents the impact of link instability and failure. We have devised a novel greedy set selection (GSS) algorithm which effectively reduces unnecessary duplications of control messages and data packets. Through extensive analysis and simulations we have shown that BRP significantly decreases the control message overhead and enhances end-to-end packet delivery ratio.

Bibliography

[1] IETF ICMP traceback working group. *http://www.ietf.org/html.charters/itrace-charter.html.*

[2] Linux 2.4 advanced routing howto: Using class based queueing for bandwidth management. *http://www.linuxdocs.org/HOWTOs/Adv-Routing-HOWTO-8.html.*

[3] Netfilter/iptables project, Netfilter: Firewalling, NAT, and packet mangling for Linux. *http://www.netfilter.org/.*

[4] OpenSSL Project, OpenSSL: The Open Source toolkit for SSL/TLS. *http://www.openssl.org/.*

[5] FIPS 180-1, Secure Hash Standard. *U.S. Department of Commerce/NIST, National Technical Information Service*, Apr. 1995.

[6] 1998 world cup web site access logs. *http://ita.ee.lbl.gov/html/contrib/WorldCup.html*, Apr. 2000.

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

[8] FIPS 197, AES: Advanced Encryption Standard. *http://csrc.nist.gov/CryptoToolkit/aes/*, Nov. 2001.

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

[10] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. *Hotnets 2003*, Nov. 2003.

[11] R. Arends and et.al. DNS security introduction and requirements. *Internet Draft, draft-ietf-dnsext-dnssec-intro-03, IETF*, Oct. 2002.

[12] G. Ateniese and S. Mangard. A new approach to DNS security (DNSSEC). *ACM Conf. on Computer and Communications Security*, 2001.

[13] S. Biswas and R. Morris. Exor: Opportunistic routing in multi-hop wireless networks. In *Proceedings of the ACM SIGCOMM '05 Conference*, Philadelphia, Pennsylvania, August 2005.

[14] A. Bremler-Barr and H. Levy. Spoofing prevention method. *in Prof. of IEEE INFOCOM*, mar. 2005.

[15] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *ACM/IEEE International Conference on Mobile Computing and Networks(MobiCom'98)*, 1998.

[16] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. *Unpublished Paper*, Dec. 1999.

[17] CERT. QUARTERLY TRENDS AND ANALYSIS REPORT. *http://www.us-cert.gov/press_room/trendsandanalysisQ406.pdf*, November 28, 2006.

[18] C. Choi, Y. Dong, and Z.-L. Zhang. Implementation of SNS authenticated packet forwarding mechanism. *Techincal Report, CS, UMN*, 2003.

[19] C. Choi, Y. Dong, and Z.-L. Zhang. LIPS: Lightweight Internet Permit System for Stopping Unwanted Packets. *In Proc. of IFIP Networking*, 2005.

[20] C. Choi, Z.-L. Zhang, and M. Srinivasan. Bubble Routing: Routing without Graphs for Multihop Wireless Networks. *Technical Report, CS, UMN*, 2007.

[21] D. Clark. The design philosophy of the DARPA Internet protocols. In *Proc. ACM SIGCOMM*, Aug. 1988.

[22] T. Cormen, C. Leiserson, and R. Rivest. Introduction to algorithm. *MIT Press, ISBN 0262031418*, 1986.

[23] D. D. Couto, D. Aguayo, J. Bicket, and R. Morris. High-throughput path metric for multi-hop wireless routing. *In MOBICOM*, Sep 2003.

[24] T. Dierks and E. Rescorla. The TLS protocol. *Internet Draft, draft-ietf-tls-rfc2246-bis-02.txt*, Oct. 2002.

[25] Y. Dong, C. Choi, and Z.-L. Zhang. An Authentication Framework for Protecting Critical Internet Services. *Journal of Microprocessors and Microsystems*, Volume 28, Issue 10 Secure Computing Platforms 1 December, 2004.

[26] Y. Dong, C. Choi, and Z.-L. Zhang. Secure Name Service: A Framework for Protecting Critical Internet Resources. *In Proc. of IFIP Networking*, 2004.

[27] Y. Dong, C. Choi, and Z.-L. Zhang. LIPS: A Lightweight Permit System for Packet Origin Accountability. *Computer Networks*, Volume 50, Issue 18, 21 December, 2006.

[28] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. *ACM MobiCom*, 2004.

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

[30] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. *Request for Comments 2827, Internet Engineering Task Force*, May 2000.

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

[32] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. In *European Symposium on Algorithms*, pages 179–193, 1996.

[33] Z. Haas, J. Halpern, and L. Li. Gossip-based ad hoc routing. *In Proc. of IEEE INFOCOM*, 2002.

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

[35] D. Harkins and D. Carrel. The internet key exchange (IKE). *RFC2409,Internet Engineering Task Force*, Nov. 1998.

[36] P. Jacquet, P. Muhlethaler, and A. Qayyum. Optimized link state routing protocol. In *IETF MANET, Internet draft*, 1998.

[37] R. K. Jain. The art of computer systems performance analysis. *John Wiley & Sons*, Apr. 1991.

[38] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*. Kluwer Academic Publishers, 1996.

[39] S. Kent and R. Atkinson. Security architecture for the internet protocol. *RFC2401, Internet Engineering Task Force*, Nov. 1998.

[40] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. *In Proc. of ACM SIGCOMM*, Aug. 2002.

[41] S. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. *In Proceedings of the IEEE ICC*, 2001.

[42] S.-J. Lee and M. Gerla. AODV-BR: Backup routing in ad hoc networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2000)*, 2000.

[43] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. Save: Source address validity enforcement. *in Prof. of IEEE INFOCOM*, 2002.

[44] H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. *In Proc. of the ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems(MSWIM)*, 2000.

[45] W. Litwin. Linear hashing: A new tool for file and table addressing. *In Proceedings of VLDB*, 1980.

[46] M. K. Marina and S. R. Das. Ad hoc on-demand multipath distance vector routing. *Wiley Wireless Communications and Mobile Computing (WCMC)*, 2001.

[47] A. Menezes, P. Oorschot, and S. Vanstone. Handbook of applied cryptography. *CRC Press, ISBN: 0-8493-8523-7*, 1996.

[48] S. Miltchev, S. Ioannidis, and A. D. Keromytis. A study of the relative costs of network security protocols. *In Proc. of the USENIX Annual Conference '2002 Freenix Track*, Jun. 2002.

[49] P. Mockapetris. Domain names - concepts and facilities. *RFC1034, Internet Engineering Task Force*, Nov. 1987.

[50] G. Montenegro and C. Castelluccia. Statistically unique and cryptographically verifiable (sucv) identifiers and addresses. *In ISOC Symposium on Network and Distributed System Security (NDSS 2002), San Diego*, 2002.

[51] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host identity protocol. *http://www.ietf.org/internet-drafts/draft-ietf-hip-base-01.txt*, Oct. 25, 2004.

[52] B. Neuman and T. Ts'o. Kerberos: An authentication service for computer network. *IEEE Comminucation Magazine*, Sept 1995.

[53] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The broad-cast storm problem in a mobile ad hoc network. In *ACM/IEEE International Con-ference on Mobile Computing and Networking (MOBICOM)*, 1999.

[54] nsnam. The network simulator. *http://nsnam.isi.edu/nsnam.*

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

[56] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. *Proc. of ACM SIGCOMM 2001, San Diago, CA.*

[57] V. D. Parka and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. IEEE INFOCOM*, 1997.

[58] W. Peng and X. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *MOBIHOC*, 2000.

[59] W. Peng and X. Lu. AHBP: An efficient broadcast protocol for mobile ad hoc networks. *Journal of Science and Technology*, 2002.

[60] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications.*

[61] C. Perkins and E. Royer. Ad hoc on demand distance vector routing, mobile computing systems and applications. In *In Proceedings of WMCSA 99*, 1999.

[62] A. Qayyum, L. Viennot, and A. Laouiti. Multipointrelaying: An efficient technique for flooding in mobile wireless networks. Technical report, 2000.

[63] 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.

[64] T. Rappaport. Wireless communications: Principles and practice. *Prentice Hall*, 1996.

[65] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection*, 1998.

[66] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). *RFC1771, Internet Engineering Task Force*, March 1995.

[67] Roofnet. Mit roofnet. *http://pdos.csail.mit.edu/roofnet.*

[68] S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson. Practical network support for IP traceback. *In Proc. of ACM SIGCOMM*, Aug. 2000.

[69] B. Schneier. Description of a network variable-length key, 64-bit block cipher (blowfish). *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Dec. 1993.

[70] W. Stalling. Cryptography and Network Security: Principles and Practice(3rd Edition). *Prentice Hall*, August 2002.

[71] R. Stone. Centertrack: An IP overlay network for tracking DoS floods. *Proc. of 2000 USENIX Secuirty Symposium*, July, 2000.

[72] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbset. Nlanr/dast: Iperf 1.7.0 - the tcp/udp bandwidth measurement tool. *http://dast.nlanr.net/Projects/Iperf/*, 2003.

[73] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer two tunneling protocol, l2tp. *RFC 2661*, August 1999.

[74] A. Valera, W. Seah, and S. Rao. Cooperative packet caching and shortest multipath routing in mobile ad hoc networks. In *IEEE INFOCOM*, 2003.

[75] H. Wang, A.Bose, M.Gendy, and K.Shin. IP Easy-pass: Edge Resource Access Control. *In Proc. of IEEE INFOCOM*, 2004.

[76] C. Westphal. Opportunistic Routing in Dynamic Ad Hoc networks. *In Proc. of IEEE MASS*, Oct. 2006.

[77] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, 2002.