

# SECURITY ARCHITECTURE AND PROTOCOLS FOR OVERLAY NETWORK SERVICES

A Thesis  
Presented to  
The Academic Faculty

by

**Mudhakar Srivatsa**

In Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy in the  
College of Computing

Georgia Institute of Technology  
August 2007

# SECURITY ARCHITECTURE AND PROTOCOLS FOR OVERLAY NETWORK SERVICES

Approved by:

Dr. Ling Liu, Advisor  
College of Computing  
*Georgia Institute of Technology*

Dr. Mustaque Ahamad  
College of Computing  
*Georgia Institute of Technology*

Dr. Wenke Lee  
College of Computing  
*Georgia Institute of Technology*

Dr. Calton Pu  
College of Computing  
*Georgia Institute of Technology*

Dr. Raghupathy Sivakumar  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Dr. Vijay Atluri  
Management Science and Information  
Systems Department  
*Rutgers University*

Date Approved: 12 April 2007

*To my family,*

*Srinivasa K. Srivatsa, Umadevi Srivatsa and Karthik K. Srivatsa.*

## ACKNOWLEDGEMENTS

This thesis would not have been possible without the generous help and encouragement of many individuals. I would like to express my gratitude to everyone who has contributed to the process leading to my dissertation. Here I would like to mention a few of these people.

First and foremost, I would like to thank my advisor Prof. Ling Liu for her tremendous help and guidance in every phase and aspect of my Ph.D. study. The flexibility and freedom she has provided me in deciding and evolving my research focus has been an invaluable asset in my quest to acquire, generate, and disseminate new knowledge. I am indebted for the countless hours she has spent on perfecting my work and the immense amount of advice she has given me on research, and career related matters.

I would like to give my special thanks to Prof. Mustaque Ahamad and Prof. Calton Pu for their timely advice. I would like to also thank every member of our DiSL research group for providing a friendly and dynamic working environment and for engaging in insightful research discussions with me, that have helped in continuously improving and polishing my work. I would like to thank my committee members Prof. Vijay Atluri, Prof. Wenke Lee, and Prof. Raghupathy Sivakumar for being very supportive of my research and for their constructive critiques that have greatly contributed to my thesis.

I would like to thank my dear friends Mohan K. Bobba and Kapil Gupta for being the colors of my life in Atlanta, and my long time friends S. Dasarathi and K. Jayanth Kumar for always keeping in touch with me. I would like to dedicate this work to my father, mother, and brother for their everlasting love and gratuitous support for me.

# TABLE OF CONTENTS

DEDICATION . . . . .	iii
ACKNOWLEDGEMENTS . . . . .	iv
LIST OF TABLES . . . . .	x
LIST OF FIGURES . . . . .	xi
SUMMARY . . . . .	xvi
I INTRODUCTION . . . . .	1
1.1 Overlay Network Computing . . . . .	2
1.1.1 Publish/Subscribe Networks . . . . .	3
1.1.2 VoIP Networks . . . . .	4
1.2 Security Issues . . . . .	5
1.2.1 Denial of Service (DoS) Attacks . . . . .	5
1.2.2 Masquerading and Spoofing Attacks . . . . .	6
1.2.3 Eavesdropping and Corruption Attacks . . . . .	6
1.2.4 User Identity Attacks . . . . .	7
1.3 Thesis Contribution . . . . .	7
1.3.1 Research Philosophy . . . . .	8
1.3.2 Information Hiding on Overlay Networks . . . . .	9
1.3.3 Secure Information Dissemination on Overlay Networks . . . . .	10
1.3.4 Privacy Beyond End-to-End Encryption in Overlay Networks . . . . .	11
1.4 Thesis Organization . . . . .	13
II OVERLAY NETWORK SECURITY . . . . .	15
2.1 Introduction . . . . .	15
2.2 Background and Terminology . . . . .	18
2.3 Threat Model . . . . .	20
2.4 Attacks on the Routing Scheme . . . . .	21
2.4.1 Alternate Lookup Paths . . . . .	22

2.4.2	Alternate Optimal (less costly) Lookup Paths . . . . .	24
2.4.3	Detecting and Recovering from Invalid Lookups . . . . .	24
2.4.4	Experimental Validation . . . . .	28
2.5	Attacking the ID-to-Key Mapping Scheme . . . . .	31
2.5.1	Quantitative Analysis . . . . .	33
2.5.2	Experimental Validation . . . . .	35
2.5.3	Defense . . . . .	35
2.6	LocationGuard . . . . .	36
2.6.1	Targeted File Attacks . . . . .	36
2.6.2	LocationGuard Approach . . . . .	37
2.6.3	Concepts and Definitions . . . . .	38
2.6.4	LocationGuard File System . . . . .	39
2.7	Location Keys . . . . .	41
2.8	Routing guard . . . . .	43
2.8.1	Overview . . . . .	44
2.8.2	Determining the Safe Obfuscation Range . . . . .	45
2.8.3	Ensuring Safe Obfuscation . . . . .	47
2.8.4	Strength of Routing guard . . . . .	48
2.9	Location Inference Guards . . . . .	51
2.9.1	Passive Inference Guards . . . . .	51
2.9.2	Host Compromise based Inference Guards . . . . .	56
2.9.3	Location Rekeying . . . . .	58
2.10	Experimental Evaluation . . . . .	58
2.10.1	Implementation-Based Experiments . . . . .	59
2.10.2	Simulation-Based Experiments . . . . .	61
2.10.3	Location Inference Guards . . . . .	64
2.11	Related Work . . . . .	67
2.12	Summary . . . . .	69

III	PUBLISH/SUBSCRIBE NETWORK SECURITY . . . . .	70
3.1	Introduction . . . . .	70
3.2	Preliminaries . . . . .	74
3.2.1	Reference Pub-Sub Model . . . . .	74
3.2.2	Threat Model . . . . .	77
3.3	EventGuard Overview . . . . .	78
3.3.1	Design Goals . . . . .	78
3.3.2	System Architecture . . . . .	80
3.4	EventGuard: Security Guards . . . . .	82
3.4.1	Tokens, Keys and Signatures . . . . .	82
3.4.2	Subscribe Guard . . . . .	85
3.4.3	Publish Guard . . . . .	87
3.4.4	Advertise Guard . . . . .	88
3.4.5	Unsubscribe Guard . . . . .	88
3.4.6	Unadvertise Guard . . . . .	89
3.4.7	Routing Guard . . . . .	90
3.4.8	Rekeying . . . . .	91
3.5	EventGuard: Scalable Key Management . . . . .	92
3.5.1	Numeric Attribute Based Matching . . . . .	93
3.5.2	Comparison with Subscriber Group Approach . . . . .	98
3.6	EventGuard: $r$ -Resilient Network Guard . . . . .	102
3.6.1	Communication Cost . . . . .	104
3.6.2	Resilience to Message Dropping Attacks . . . . .	105
3.6.3	Low Cost Resilient Networks . . . . .	106
3.7	EventGuard Evaluation . . . . .	108
3.7.1	Micro-Benchmarks . . . . .	108
3.7.2	Key Management . . . . .	112
3.7.3	Macro-Benchmarks . . . . .	113
3.8	Related Work . . . . .	120

3.9	Summary . . . . .	122
IV	VOIP NETWORK SECURITY . . . . .	123
4.1	Introduction . . . . .	123
4.2	Preliminaries: Skype Lookup Protocol . . . . .	125
4.3	Attacks on the Session Initiation Protocol . . . . .	127
4.3.1	Caller Identification Attacks . . . . .	128
4.3.2	Countering Triangulation based Timing Attacks . . . . .	137
4.3.3	Experimental Evaluation . . . . .	144
4.4	Attacks on Voice Session . . . . .	148
4.4.1	Flow Analysis Attacks . . . . .	150
4.4.2	VoIP Privacy using $k$ -Anonymity . . . . .	158
4.4.3	Experimental Evaluation . . . . .	165
4.5	Related Work . . . . .	167
4.6	Summary . . . . .	168
V	TRUST AND REPUTATION MANAGEMENT . . . . .	169
5.1	Introduction . . . . .	169
5.2	TrustGuard: An Overview . . . . .	171
5.2.1	System Architecture . . . . .	171
5.2.2	Problem Statement and Solution Approach . . . . .	173
5.3	Strategic Malicious Nodes . . . . .	174
5.3.1	Cost Model . . . . .	175
5.3.2	Dependable Trust Model . . . . .	176
5.3.3	Fading Memories . . . . .	179
5.4	Fake Transactions . . . . .	181
5.4.1	Unforgeable Transaction Proofs . . . . .	183
5.4.2	Fair Exchange of Transaction Proofs . . . . .	183
5.5	Dishonest Feedbacks . . . . .	185
5.6	Performance Enhancements . . . . .	188

5.6.1	Efficiently Storing Feedbacks . . . . .	188
5.6.2	Minimizing the Replica Lookup Cost . . . . .	189
5.7	Evaluation . . . . .	191
5.7.1	Guarding from Strategic Node Behaviors . . . . .	191
5.7.2	Guarding from Fake Transactions . . . . .	196
5.7.3	Guarding from Dishonest Feedbacks . . . . .	197
5.7.4	Transaction Success Rate . . . . .	198
5.8	Discussion . . . . .	199
5.8.1	Issues . . . . .	199
5.8.2	Related Work . . . . .	200
5.9	Summary . . . . .	201
VI	CONCLUSIONS AND FUTURE WORK . . . . .	203
6.1	Open Issues and Future Research Directions . . . . .	206
6.1.1	Open Issues in the Context of this Thesis . . . . .	206
6.1.2	An Outlook for Future Research Directions . . . . .	208
	REFERENCES . . . . .	210
	VITA . . . . .	218

## LIST OF TABLES

1	Probability of Lookup Failure ( $p = 10\%$ ): Quantitative Analysis . . .	29
2	Attack on ID Mapping Scheme . . . . .	35
3	Lookup Identifier obfuscation . . . . .	44
4	LocationGuard File Types . . . . .	59
5	Mean Fraction of Good Nodes in Uncompromised State ( $G'$ ) . . . . .	63
6	Time Interval between Location ReKeying (normalized by $\frac{1}{\lambda}$ time units)	64
7	Entropy (in number of bits) of a Zipf-distribution . . . . .	64
8	Countering Lookup Frequency Inference Attack Approach I: Result Caching (with 32K files) . . . . .	64
9	Max Cost . . . . .	98
10	Avg Cost: $R = 10^4$ . . . . .	98
11	KDC Costs . . . . .	102
12	Subscriber Costs . . . . .	102
13	Theoretical Lower Bound: $NS = 10^3$ and $R = 10^4$ . . . . .	103
14	Theoretical Lower Bound: $\phi_R = 100$ and $R = 10^4$ . . . . .	103
15	Computation Overheads for EventGuard Operations: $w$ is some topic, $pbl$ is a publication, and $m$ denotes the number of topics marked on message $pbl$ . . . . .	108
16	Message Size Overhead due to EventGuard including only those mes- sages sent on the pub-sub network: $m$ denotes the number of topics marked on the publication . . . . .	109
17	EventGuard Storage Overhead: $HT_{size}$ denotes the total size of the hashtable maintained for detecting flooding based DoS attacks ( $HT_{size}$ is at most a few tens of KBs) . . . . .	109
18	Relative Cost paid by Malicious Nodes Vs $T_{off}$ (normalized by $maxH$ )	197

## LIST OF FIGURES

1	Lookup algorithm for legitimate nodes . . . . .	25
2	Probability of a lookup failure: Initial $TTL = 100$ and $p = 10\%$ . . . . .	29
3	Lookup Costs: Scenario 1 with $M = 5$ . . . . .	30
4	Lookup Cost: Scenario 2 with $M = 5$ . . . . .	30
5	ID Mapping scheme: attack a specific data item $d$ . . . . .	33
6	LocationGuard: System Architecture . . . . .	37
7	LocationGuard: Conceptual Design . . . . .	37
8	Lookup Using File Identifier Obfuscation: Illustration . . . . .	44
9	Countering Frequency Analysis Attack by file identifier obfuscation. $X_1X_2$ , $Y_1Y_2$ and $Z_1Z_2$ denote the ranges of the obfuscated identifiers of files $f_1, f_2, f_3$ stored at node $n$ . Frequency inference attacks works in scenario (i), but not in scenario (ii). Given an identifier $otk \in Y_1Z_1$ , it is hard for an adversary to guess whether the lookup was for file $f_1$ or $f_2$ . . . . .	55
10	Implementation Architecture . . . . .	59
11	File Read Overhead . . . . .	59
12	File Write Overhead . . . . .	59
13	Probability of a Target File Attack for $N = 1024$ nodes and $R = 7$ using DoS Attack . . . . .	62
14	Probability of a Target File Attack for $N = 1024$ nodes and $R = 7$ using Host Compromise Attack (with no token collection) . . . . .	62
15	Probability of a Target File Attack for $N = 1024$ nodes and $R = 7$ using Host Compromise Attack with token collection from compromised nodes . . . . .	62
16	Countering Lookup Frequency Inference Attack Approach II: File Identifier obfuscation . . . . .	65
17	Countering File Replica Frequency Inference Attack: Location Rekeying Frequency Vs File Update Frequency . . . . .	65
18	Basic Pub-Sub System . . . . .	74
19	EventGuard Architecture . . . . .	74
20	Handling Flooding based DoS attacks in EventGuard . . . . .	91

21	Constructing Resilient Networks: Thick lines represent links in the binary tree network and the dashed lines represent additional links added to binary tree network to make its $ind = 2$ . . . . .	91
22	Key Tree: Range Queries on Numeric Attributes . . . . .	103
23	Num Keys per Subscriber . . . . .	112
24	Num Keys per Publisher . . . . .	112
25	KDC Load . . . . .	112
26	Confidentiality and Integrity . . . . .	114
27	Flooding-based DoS Attack . . . . .	114
28	$MS$ Load . . . . .	114
29	Communication Cost Vs Number of Receptients $N(w)$ . . . . .	116
30	Resilience Vs $a$ with $ind = 1$ . . . . .	116
31	Resilience Vs $ind$ with $a = 6$ . . . . .	116
32	Throughput . . . . .	118
33	Latency . . . . .	118
34	Resilience to Flooding-based DoS Attacks . . . . .	118
35	Skype Peer-to-Peer VoIP Network . . . . .	126
36	Shortest Path using Skype P2P Lookup Protocol . . . . .	126
37	Triangulation Attack Illustration: Caller Lies in Shaded Region . . . . .	126
38	Distance Distribution . . . . .	132
39	Caller Identification with $\epsilon = 10ms$ . . . . .	132
40	Caller Identification with 10 Malicious Nodes . . . . .	132
41	Stochastic Shortest Path Algorithm . . . . .	135
42	Stochastic Vs Deterministic Triangulation Attack . . . . .	135
43	Top-10 Probability . . . . .	135
44	10 Malicious Nodes . . . . .	135
45	Differential Vs Stochastic Triangulation Attack . . . . .	137
46	Top-10 Probability . . . . .	137
47	10 Malicious Nodes . . . . .	137

48	Top- $\kappa$ Probability . . . . .	142
49	Number of Malicious Nodes . . . . .	142
50	Random Walk Search Algorithm . . . . .	142
51	Latency Perturbation . . . . .	146
52	Controlled Random Walk . . . . .	146
53	Multi-Agent Random Walk . . . . .	146
54	Latency Perturbation . . . . .	146
55	Controlled Random Walk . . . . .	146
56	Multi-Agent Random Walk . . . . .	146
57	Optimal Parameter Setting . . . . .	147
58	Top-10 Probability . . . . .	147
59	Top-k . . . . .	147
60	VoIP Flow Bandwidth: G.729A (CS-CELP) Audio Codec with 8 Kbps Compression and Packet Duration 20ms . . . . .	150
61	Flow Analysis Attack . . . . .	150
62	Mixing Flows . . . . .	150
63	Resisting Flow Analysis Attack . . . . .	150
64	Tracing Voice Calls . . . . .	150
65	Shortest Path Tracing . . . . .	150
66	Tracing Algorithm . . . . .	151
67	Shortest Path Tracing Algorithm . . . . .	152
68	Shortest Path Tracing Vs Naive Tracing . . . . .	153
69	Precision and Recall with 128 Erlang Call Volume . . . . .	153
70	$F$ -measure with $\kappa = 2$ . . . . .	153
71	Entropy with $\kappa = 2$ . . . . .	156
72	Top- $m$ Probability with $\kappa = 2$ and Latency Prior . . . . .	156
73	Computation Cost . . . . .	156
74	Compromised Proxies . . . . .	159
75	1-anonymity . . . . .	159

76	2-anonymity . . . . .	159
77	AASIP Vs SIP: Top-10 Probability . . . . .	163
78	AASIP Vs SIP: 128 Erlangs . . . . .	163
79	AASIP Vs SIP: Path Latency . . . . .	163
80	Compromised Proxies . . . . .	164
81	Computation Cost: $k$ -Anonymity . . . . .	164
82	Computation Cost: Tolerating Malicious Proxies . . . . .	164
83	Messaging Cost . . . . .	165
84	Search Latency . . . . .	165
85	Node Load . . . . .	165
86	TrustGuard Architecture . . . . .	171
87	Cost of Building Reputation . . . . .	176
88	Dependable Trust Model: Illustration . . . . .	177
89	Updating Fading Memories: $FTV[i]$ denotes the faded values at time $t$ and $FTV'[i]$ denotes the faded values at time $t + 1$ . . . . .	180
90	Cost of Building Reputation with Delayed Conflict Resolution . . . . .	185
91	Model I . . . . .	188
92	Optimistic versus Pessimistic Summarization . . . . .	188
93	Effect of Varying Parameters in the Trust Model . . . . .	188
94	Probability of False Positives and False Negatives with 100% Collusion	190
95	Probability of False Positives and False Negatives with 20% Collusion	190
96	Trust Value Lookup Cost . . . . .	190
97	Trust Model with a Small History . . . . .	193
98	Trust Model with a Large History . . . . .	193
99	Trust Model with Fading Memories . . . . .	193
100	Fair Exchange of Transaction Proofs: Optimistic Vs Trust-Value Based Exchange Protocol . . . . .	195
101	Robustness in Non-Collusive Setting . . . . .	195
102	Robustness in Collusive Setting . . . . .	195

103	Transaction Success Rate: Assuming No Fake or Dishonest Feedbacks	198
104	Transaction Success Rate: Non-Collusive and Collusive Settings . . .	198

## SUMMARY

The recent years have witnessed a wide spread proliferation of the Internet to encompass multiple autonomous organizations, heterogeneous platforms and mobile and wireless computing stations. This ubiquitous nature of the Internet not only makes it easy to access information and services from anywhere at anytime, but also facilitates the acquisition and generation of new information. The ever growing pervasiveness of the Internet has been accompanied by an increasing number of security threats. We have witnessed several security attacks against online services either for extortion reasons or for impairing and even disabling the competition. Therefore, we believe that security is becoming an increasingly integral component of a system's correctness.

The wide spread growth of the Internet is best captured by the emergence of the overlay network computing systems. Overlay network computing systems and applications have continued to evolve over the past decade, ranging from SETI@Home and music sharing systems (Gnutella, KaZaa, and Limewire) to more sophisticated applications, including file storage systems (cooperative file system (CFS), Farsite, and OceanStore), publish-subscribe systems (Siena, Scribe, and Gryphon), and Skype-like voice over IP (VoIP) systems. The overlay network computing model provides many opportunities for information dissemination across different organizational boundaries, heterogeneous platforms, and a large, dynamic population of users and has shown the potential to become a prominent network computing paradigm for large scale distributed applications. The massive distributed nature of these applications exposes them to a wide range of security threats ranging from eaves dropping, data corruption and denial of service (DoS) attacks.

Conventional wisdom suggests that in order to build a secure system, security must be an integral component in the system design. However, cost considerations drive most system designers to channel their efforts on the system's performance, scalability and usability. With little or no emphasis on security, such systems are vulnerable to a wide range of attacks that can potentially compromise confidentiality, integrity and availability of sensitive data. It is often cumbersome to redesign and implement massive systems with security as one of the primary design goals. This thesis advocates a proactive approach that cleanly retrofits security solutions into existing system architectures. The first step in this approach is to identify security threats, vulnerabilities and potential attacks on a system or an application. The second step is to develop security tools in the form of customizable and configurable plug-ins that address these security issues and minimally modify existing system code, while preserving its performance and scalability metrics.

This thesis demonstrates techniques to build secure frameworks for massively distributed applications in a way that preserves the application's performance, scalability, ease of use, cost effectiveness and real-time guarantees. We use overlay network applications to shepherd through and address challenges involved in supporting security in large scale distributed systems. In particular, the focus is on two popular applications: publish/subscribe networks and VoIP networks. Briefly, the following contributions are made in the areas of secure overlay network applications:

- Our work on VoIP networks has for the first time identified and formalized caller identification attacks on VoIP networks. We have identified two attacks: a triangulation based timing attack on the VoIP network's route set up protocol and a flow analysis attack on the VoIP network's voice session protocol. These attacks allow an external observer (adversary) to *uniquely* (nearly) identify the *true* caller (and receiver) with high probability [103, 101] (see chapter 4).
- Our work on the publish/subscribe networks has resulted in the development of

an unified framework for handling event confidentiality, integrity, access control and DoS attacks, while incurring small (6%) overhead on the system. Our work has for the first time proposed a key isomorphism paradigm to preserve the confidentiality of events on publish/subscribe networks while permitting scalable content-based matching and routing [98, 102] (see chapter 3).

- Our work on overlay network security has resulted in a novel information hiding technique on overlay networks. Our solution represents the first attempt to transparently hide the location of data items on an overlay network [99, 97] (see Chapter 2).
- Our work on overlay network security has identified several vulnerabilities in trust management systems. We have shown that strategic malicious behavior can adversely affect the efficacy of a trust management system [100, 104] (see chapter 5).

# CHAPTER I

## INTRODUCTION

The recent years have witnessed a wide spread proliferation of the Internet to encompass multiple autonomous organizations, heterogeneous platforms and mobile and wireless computing stations. This ubiquitous nature of the Internet not only makes it easy to access information and services from anywhere at anytime, but also facilitates the acquisition and generation of new information. The ever growing pervasiveness of the Internet has been accompanied by an increasing number of security threats. We have witnessed several security attacks against online services either for extortion reasons or for impairing and even disabling the competition. Therefore, we believe that security is becoming an increasing integral component of a system's correctness.

With the wide spread growth of the Internet, overlay networks have emerged as a new parallel and distributed computing paradigm. Overlay network computing systems and applications have continued to evolve over the past decade, ranging from SETI@Home and music sharing systems (Gnutella [38], KaZaa [50], and Limewire [56]) to more sophisticated applications, including file storage systems (cooperative file system (CFS) [23], Farsite [7], and OceanStore [53]), publish-subscribe systems (Siena [16], Scribe [25], and Gryphon [12]), and Skype-like voice over IP (VoIP) systems [2]. The overlay network computing model provides many opportunities for information dissemination across different organizational boundaries, heterogeneous platforms, and a large, dynamic population of users and has shown the potential to become a prominent network computing paradigm for massively distributed applications.

Conventional wisdom suggests that in order to build a secure system, security must be an integral component in the system design. However, cost considerations

drive most system designers to channel their efforts on the system's performance, scalability and usability. With little or no emphasis on security, such systems are vulnerable to a wide range of attacks that can potentially compromise confidentiality, integrity and availability of sensitive data. It is often cumbersome to redesign and implement massive systems with security as one of the primary design goals. This thesis advocates a proactive approach that cleanly retrofits security solutions into existing system architectures. The first step in this approach is to identify security threats, vulnerabilities and potential attacks on a system or an application. The second step is to develop security tools in the form of customizable and configurable plug-ins that address these security issues and minimally modify existing system code, while preserving its performance and scalability metrics.

Before discussing the security challenges in overlay network applications, we provide some background information on overlay network computing. We use two popular overlay network applications: publish/subscribe networks and VoIP networks to demonstrate their rich functionality, ease of deployment and use, cost effectiveness, real-time guarantees, and performance and scalability benefits.

### ***1.1 Overlay Network Computing***

An overlay network is a virtual network created on top of an existing transport network such as TCP/IP. Unlike traditional distributed computing, overlay networks aggregate large number of computers and possibly mobile or hand-held devices. An overlay network provides mechanisms to create and maintain the connectivity of an individual node to the network by establishing network connections with a subset of other nodes (neighbors) in the overlay network. The overlay network employs routing protocols that allow individual computers and devices to share information and resources directly, thereby obviating the need for dedicated servers. Overlay networking technologies provide some desirable system properties for supporting pervasive

and cooperative application sharing across the Internet, such as anonymity, fault tolerance, low maintenance & low administration cost, and transparent and dynamic operability.

### **1.1.1 Publish/Subscribe Networks**

A growing number of emerging Internet applications requires information dissemination across different organizational boundaries, heterogeneous platforms, and a large, dynamic population of publishers and subscribers. A publish-subscribe overlay service is a wide-area communication infrastructure that enables data access and data sharing across potentially unlimited number of publishers and subscribers, scattered geographically across the wired and wireless Internet. A wide-area publish-subscribe (pub-sub for short) system is often implemented as a collection of spatially disparate computing nodes (or network servers) communicating with each other through content-based routing protocols on top of an overlay network. In such an environment, publishers publish information in the form of event notifications and subscribers have the ability to express their interests in an event or a pattern of events in the form of subscription constraints. The pub-sub overlay network uses content-based routing schemes to dynamically match each publication against all the active subscriptions, and notifies the subscribers of any publication that matches their registered interest, ensuring that subscribers only receive notifications of those events that match their subscribed interests.

An important characteristic of pub-sub overlay services is the decoupling of publishers and subscribers combined with content-based routing protocols, enabling a many (publishers) to many (subscribers) communication model. Such a model presents many inherent benefits. By offloading the task of identifying destination addresses of publication events from the publishers to the pub-sub overlay network, it not only

allows message routing to be handled in a way that avoids unnecessary message replications but also enables dynamic and fine-grained subscriptions. As a result, the pub-sub overlay services have proven to be scalable and effective for wide-area information dissemination across distinct administrative domains and heterogeneous systems.

### **1.1.2 VoIP Networks**

Voice over IP (VoIP), also known as Internet telephony or IP telephony, is emerging as a mature and practical technology alternative to traditional public switched telephone networks (PSTN). VoIP technology enables people to make phone calls through public Internet. As audio quality, bandwidth usage, and setup convenience are reaching acceptable levels, many organizations have switched to VoIP. According to TeleGeography Research, world wide VoIP's share of voice traffic has grown from 12.8% in 2003 to an estimated 75% in 2007. There are three important reasons for this wide spread adoption of VoIP networks. First, one of the most prevailing reasons for such rapid VoIP deployment is reduced costs. Second, VoIP offers very rich call forwarding features including, redirecting a call based on the time of the day, etc. Another indisputable reason is the fact that VoIP represents a significant step towards the integration of voice and data networks.

A VoIP overlay service is a wide-area communication infrastructure that enables phone calls between callers and receivers that are scattered geographically across the wired & wireless Internet, landlines, cell phones, etc. A VoIP network is implemented as a collection of spatially disparate computing nodes (VoIP proxy servers) communicating with each other through two routing protocols on top of a peer to peer overlay network. The SIP (session initiation protocol) sets up a voice path between the caller and the receiver. The voice path includes one or more nodes on the VoIP overlay network such that two consecutive nodes on the path are neighbors on the overlay

network. The RTP (real-time transport protocol) is used for exchange voice packets between the caller and the receiver. The voice traffic is itself encoded (and encrypted) using standard audio codec such as CS-CELP. Similar to the pub-sub overlay network, a VoIP network achieves significant performance and scalability benefits by offloading the task of identifying receiver and transporting the voice traffic to a large and distributed VoIP overlay network.

## **1.2 Security Issues**

### **1.2.1 Denial of Service (DoS) Attacks**

Recently we have seen increasing numbers of denial of service (DoS) attacks against online services and web applications either for extortion reasons, or for impairing and even disabling the competition [18, 55, 73]. These DoS attacks are increasingly mounted by professional attackers using huge zombie networks consisting of thousands of compromised machines on the Internet [43, 108, 105]. An FBI affidavit [18] describes a DoS attack on an e-Commerce website using a 5,000 node zombie net that caused a loss of several millions of dollars in revenue. Hence, countering DoS attacks on online services has become a very challenging problem.

The overlay network has to protect the applications data hosted by the overlay nodes from DoS and host compromise attacks. Protecting the overlay network nodes from DoS and host compromise attacks improves the availability of an application. In an overlay network model, DoS attacks can target three different layers: (i) TCP/IP layer [87, 35, 13, 113], (ii) overlay network layer [17, 51], and the application layer [49, 115]. In addition to attacks from external adversaries, the overlay network has to develop solutions to mitigate *insider* DoS attacks, wherein a set of malicious overlay nodes attempt to launch a DoS attack on the applications hosted by the overlay network.

### 1.2.2 Masquerading and Spoofing Attacks

The overlay network has to protect the applications data hosted by the overlay nodes from incorrect or fake (spoofed) application data. Protecting the overlay network nodes from incorrect or fake application data guarantees the authenticity of application data hosted by the nodes. In an overlay network, authenticity attacks can be of two types: (i) an adversary may attempt to spoof the identity of a legitimate content provider and send incorrect or fake application data to the overlay network nodes [109], and (ii) an authentic content provider may flood the overlay network nodes with incorrect or inaccurate application data [119, 48]. The latter problem is prevalent in today's Internet wherein, we have multiple competitive web servers (with possibly conflicting interests) publishing doctored information [42].

### 1.2.3 Eavesdropping and Corruption Attacks

The overlay network has to protect the confidentiality and integrity from: (i) the overlay network nodes, and (ii) unauthorized users. In several applications, the clients may not trust the overlay network nodes with the confidentiality and integrity of the application data. The malicious overlay network nodes may be able to eavesdrop or corrupt the application data hosted by them. In addition, malicious overlay nodes may collude with one another in their attempts to compromise the confidentiality and integrity of application data.

The overlay network allows users to specify access control rules on application data. These access control rules restrict the set users that can access a given piece of application data hosted by the overlay network. However, malicious users may be curious to access application data and services that they are not authorized to access. In addition, malicious users may collude with one another and with the malicious nodes in the overlay network to compromise the confidentiality and integrity of application data.

### 1.2.4 User Identity Attacks

In several user privacy sensitive applications such VoIP network, an overlay network has to protect the identity of the users from an external observer (adversary). The overlay network in the VoIP scenario has to not only maintain the voice traffic a secret, but also hide the 'who is talking to whom' information. The use of VoIP overlay network has made it much easier to achieve anonymity in voice communications, especially when VoIP calls are made between computers. Since VoIP calls between peer computers have no phone numbers associated with them, and they could easily be protected by end-to-end encryption and routed through low latency anonymizing networks (e.g., Onion Routing [40], Tor [28], Freedom [11], and Tarzan [37]) to achieve anonymity. Intuitively, computer to computer VoIP calls could remain anonymous if they are encrypted end-to-end and routed through a low latency anonymizing network. However, the malicious overlay network nodes and the external observer may be able to observe the traffic routed through them. In addition, malicious overlay network nodes may collude with an adversary to compromise user privacy.

### 1.3 Thesis Contribution

This thesis aims at developing system-level security algorithms and techniques for several existing computing platforms and applications, focusing on overlay networks, publish/subscribe networks and VoIP networks. While it is commonly believed that it is very hard to secure large scale open distributed systems, this dissertation work presents pragmatic techniques for supporting security mechanisms that minimally modify an existing system code and yet preserve its performance and scalability metrics.

There are three main problems addressed in this thesis.

- It is widely acknowledged that using weak (weak communication and computation power) overlay network nodes, makes it more vulnerable to targeted denial

of service (DoS) attacks. We show that even when an attacker is significantly stronger than an overlay network node, one can mitigate targeted DoS attacks by hiding critical information on a large overlay network.

- It has been shown that secure and scalable information dissemination on overlay networks reduces to secure multi-party computation problems (which incur very high computation and communication costs). We point out that for several practical publication-subscription matching operators, one can indeed achieve both scalable and secure information dissemination protocols on overlay networks.
- It is widely believed that using end-to-end encryption on overlay networks applications, such as voice and multi-media streaming, protects user identity. We point out that there are several other vulnerabilities that should be formally analyzed in order to effectively defend against user identity attacks on overlay networks.

### 1.3.1 Research Philosophy

The research philosophy adopted in this thesis is the notion of *retrofitting* security into legacy applications. The *input* is a large scale distributed system that has typically been tuned for functionality, performance and scalability. The first step involves studying the system using various theoretical methodologies (including network flows, statistical inference, applied cryptography, information hiding, game theory) with the goal of identifying potential security flaws in the application. The second step involves developing systems-level security solutions in the form of customizable plug-ins that could be neatly weaved into the original system code by minimally modifying the code itself. The third step is to perform performance and scalability benchmarks on the secure version of the legacy application and quantify the overhead of our security plug-ins.

The main contributions from this thesis are summarized below.

### 1.3.2 Information Hiding on Overlay Networks

We have presented techniques that exploit the structure of a distributed hash table (DHT) based overlay network to improve its resilience to DoS attacks [97]. We improve the resilience of overlay networks to DoS attacks using three techniques: replicating application data, redundant routing, and verifiable routing. Replicating application data ensures that the data is available to users as long as a threshold number of replicas of the application data are available. Redundant routing using independent paths ensures that two nodes can communicate with one another even if some of the nodes are not operational either due to DoS attacks or host compromise attacks. Verifiable routing mechanisms use DHT structures to probabilistically detect malicious nodes attempting to misguide the routing algorithm; on detecting a malicious behavior, an overlay node reroutes the message via an alternate path on the overlay network.

We have proposed a location hiding algorithm that can hide the location of application data or an online service on a large overlay network [99]. Location of data on an overlay network refers to the IP address of the overlay network node that hosts the service. The key intuition here is that without knowing the location of a service, it is very hard for an adversary to launch a DoS attack on the service. The location hiding algorithm introduces the concept of a location key. Analogous to a cryptographic key, a user can locate any named data object on an overlay network if and only if the user knows the location key associated with that data object. We exploit the structure of the overlay network to obfuscate lookup queries on the overlay network. The obfuscation methodology uses a probabilistic one-sided error algorithm that preserves the performance and scalability of the lookup protocol, while making it hard for an adversary to guess a location identifier even after observing polynomial

number of routing queries on the overlay network (polynomial in the number of bits of the location key). We have also studied inference attacks on our location hiding algorithm and proposed location re-keying based techniques to defend against them.

Several research papers have proposed reputation management systems to condone content providers providing incorrect or inaccurate application data or services [119, 48]. Reputation management systems essentially create a feedback loop that allow content consumers (users) to rate the content providers based on the quality of the data obtained from the content providers (via the overlay network service provider). In course of time, all legitimate users will rely only on the data provided by reputed content providers; hence, low quality content providers would eventually run out of business. We have identified three important vulnerabilities in reputation management systems. First, we provide a dependable trust model and a set of formal methods to handle strategic malicious nodes that continuously change their behavior to gain unfair advantages in the system. Second, a transaction based reputation system must cope with the vulnerability that malicious nodes may misuse the system by flooding feedbacks with fake transactions. Third, but not the least, we identify the importance of filtering out dishonest feedbacks when computing reputation-based trust of a node, including the feedbacks filed by malicious nodes through collusion. We have built three security guards to defend against these attacks using cryptographic techniques and Byzantine fault-tolerant techniques [100, 104, 69].

### **1.3.3 Secure Information Dissemination on Overlay Networks**

We have developed techniques to protect the confidentiality, integrity and availability of application data from the overlay network nodes for publish-subscribe networks [98]. We use cryptographic techniques adapted using application specific knowledge (pub-sub matching operators) to secure the pub-sub system. We developed six guards

(security wrappers) around six publish-subscribe primitives: publish, subscribe, advertise, unsubscribe, unadvertise and route to protect the publish-subscribe service against confidentiality & integrity attacks, and authenticity attacks. We have also developed a  $r$ -resilient publish-subscribe overlay network topology that is resilient to overlay network level DoS attacks.

We have developed techniques to support flexible access control rules on the application data provided by the content providers [102]. In a publish-subscribe service access control rules are specified as constraints. For example, a subscriber  $S$  who has subscribed for a subscription filter  $\phi = \langle\langle\text{topic}, EQ, \text{cancerTrail}\rangle, \langle\text{age}, >, 20\rangle\rangle$  should be able to read the patient record  $rec$  in an event  $e = \langle\langle\text{topic}, \text{cancerTrail}\rangle, \langle\text{age}, 25\rangle, \langle\text{patientRecord}, rec\rangle\rangle$ , but not the patient record  $rec'$  in an event  $e' = \langle\langle\text{topic}, \text{cancerTrail}\rangle, \langle\text{age}, 15\rangle, \langle\text{patientRecord}, rec'\rangle\rangle$ . We have proposed and developed the notion of *key isomorphism* to handle this problem: we associate an authorization key  $K(\phi)$  with a subscription  $\phi$  and an encryption key  $K(e)$  with event  $e$  such that  $K(e)$  is efficiently derivable from  $K(\phi)$  if and only if  $e$  matches  $\phi$ . We use the semantics of operators like  $<$ ,  $>$ ,  $\geq$ ,  $\leq$ , *substring*, *superstring*, *prefix*, and *suffix* to construct isomorphic key spaces in an efficient and scalable manner.

#### 1.3.4 Privacy Beyond End-to-End Encryption in Overlay Networks

We have identified two attacks on VoIP networks and proposed solutions to mitigate them. The first attack operates on the VoIP route set up protocol with the goal of compromising the identity of the caller [103]. We have developed three triangulation-based timing analysis attacks on the route set up protocol that exploit its broadcast nature and its shortest path properties to identify the caller with high probability. We have shown that any distributed shortest protocol is vulnerable to such triangulation-based timing analysis attacks. We exploit the fact that in a VoIP network, a one-way latency of up to 250ms is nearly unperceivable to human users. Hence, we develop

three security guards that alleviate caller identification attacks and yet satisfy the one-way latency constraint of 250ms. Our caller identification guards implement two key ideas. First, we introduce uncertainty into timing information by adding stochastic perturbations to the network latencies. Second, we propose two search algorithms that combine random walk and broadcast algorithms with the goal of reducing lookup latency and yet providing better protection against triangulation based timing attacks.

The second attack operates on the voice session. We have developed flow analysis attacks on VoIP networks [101]. A flow (sufficiently long lasting packet stream) analysis attack measures the volume of flow (packets per unit time) between two nodes on the VoIP network and uses this information in conjunction with the VoIP network topology to determine a caller & recipient pair. The constant packet rate of VoIP flows makes it easier for an external observer to trace a flow from a caller to a receiver. We describe three statistical inference based flow analysis attacks on VoIP networks with increasing sophistication. These attacks help an adversary to identify a small list of potential recipients for a given VoIP call. We have also developed practical techniques to achieve quantifiable and customizable privacy on VoIP networks. We reduce the efficacy of flow analysis attacks by mixing one or more VoIP flows. The constant packet rate nature of VoIP places makes it easy for one or more VoIP flows to be mixed without leaking much information to an external observer. We have implemented a route set up and maintenance protocol that accepts a customizable anonymity parameter  $k$  from the caller. The protocol ensures that it sets up a voice path that mixes at least  $k$ -flows while keeping the one-way path latency smaller than 250ms.

## ***1.4 Thesis Organization***

The rest of this is organized as chapters dedicated to security issues in overlay networks and two popular overlay network based application: publish/subscribe networks and VoIP networks. In each of these chapters background information including system and threat models are provided before the core technical content is described. The specific contributions are given in the introduction part of each chapter, whereas the related work in the literature is reported at the end of the chapter. Concretely, this thesis is composed of the following chapters.

Chapter 2 presents secure routing issues in large scale overlay networks. We first identify three vulnerabilities in routing protocols on large overlay networks. We advocate information hiding techniques to hide data on an overlay network such that only an authorized user can locate the data. We present modifications to the overlay network lookup and routing protocol, a formal security analysis of the protocol, and present experimental results that demonstrate the performance and scalability of the proposed protocol.

Chapter 3 presents security issues in publish/subscribe networks. We present a suite of guards to secure a publish/subscribe against a wide range of availability, confidentiality and integrity attacks. We propose the notion of key isomorphism to support event confidentiality & integrity and efficient multi-path event dissemination trees for scaleable and secure distribution of events on the publish/subscribe networks.

Chapter 4 presents security issues in VoIP networks. We identify and formally study two attacks on VoIP networks: a triangulation based timing attack that operates on the route step up protocol and a flow analysis attack that operates on the voice session. We present modifications to the VoIP network protocols, a formal statistical inference based security analysis, and experimental results that demonstrate the performance, scalability and real-time guarantees offered by the proposed protocol.

Chapter 5 presents authenticity issues in large scale overlay networks. We identify

three vulnerabilities in reputation based management systems and propose solutions to defend effectively against them. We also present experimental results that demonstrate the efficacy of our proposal against a wide range of attack strategies.

Chapter 6 discusses some related issues and concludes the thesis.

## CHAPTER II

### OVERLAY NETWORK SECURITY

A number of recent applications have been built on distributed hash tables (DHTs) based overlay networks. Almost all DHT-based schemes employ a tight deterministic data placement and ID mapping schemes. This feature on one hand provides assurance on location of data if it exists, within a bounded number of hops, and on the other hand, opens doors for malicious nodes to lodge attacks that can potentially thwart the functionality of the overlay network.

This chapter studies several serious security threats in DHT-based systems through two targeted attacks at the overlay network’s protocol layer. The first attack explores the routing anomalies that can be caused by malicious nodes returning incorrect lookup routes. The second attack targets the ID mapping scheme. We disclose that the malicious nodes can target any specific data item in the system; and corrupt/modify the data item to its favor. Third, we identify targeted file attacks, wherein an adversary attempts to attack a small (chosen) set of files (data items) by attacking the nodes that host them. For each of these attacks, we provide quantitative analysis to estimate the extent of damage that can be caused by the attack; followed by experimental validation and defenses to guard the overlay networks from such attacks.

#### *2.1 Introduction*

A new breed of serverless file storage services, like CFS [23], Farsite [7], OceanStore [53] and SiRiUS [39], have recently emerged. In contrast to traditional file systems, they harness the resources available at desktop workstations that are distributed over a wide-area network. The collective resources available at these desktop workstations amount to several peta-flops of computing power and several hundred peta-bytes of storage space [7].

These emerging trends have motivated serverless file storage as one of the most popular

application over decentralized overlay networks. An overlay network is a virtual network formed by nodes (desktop workstations) on top of an existing TCP/IP-network. Overlay networks typically support a lookup protocol. A lookup operation identifies the location of a file given its filename. Location of a file denotes the IP-address of the node that currently hosts the file. There are four important issues that need to be addressed to enable wide deployment of serverless file systems for mission critical applications.

**Efficiency of the lookup protocol.** There are two kinds of lookup protocol that have been commonly deployed: the Gnutella-like broadcast based lookup protocols [38] and the distributed hash table (DHT) based lookup protocols [106] [77] [84]. File systems like CFS, Farsite and OceanStore use DHT-based lookup protocols because of their ability to locate any file in a small and bounded number of hops.

**Malicious and unreliable nodes.** Serverless file storage services are faced with the challenge of having to harness the collective resources of loosely coupled, insecure, and unreliable machines to provide a secure, and reliable file-storage service. To complicate matters further, some of the nodes in the overlay network could be malicious. CFS employs cryptographic techniques to maintain file data confidentiality and integrity. Farsite permits file write and update operations by using a Byzantine fault-tolerant group of meta-data servers (directory service). Both CFS and Farsite use replication as a technique to provide higher fault-tolerance and availability.

**Targeted File Attacks.** A major drawback with serverless file systems like CFS, Farsite and OceanStore is that they are vulnerable to targeted attacks on files. In a targeted attack, an adversary is interested in compromising a small set of target files through a DoS attack or a host compromise attack. A denial-of-service attack would render the target file unavailable; a host compromise attack could corrupt all the replicas of a file thereby effectively wiping out the target file from the file system. The fundamental problem with these systems is that: (i) the number of replicas ( $R$ ) maintained by the system is usually much smaller than the number of malicious nodes ( $B$ ), and (ii) the replicas of a file are stored at *publicly known* locations. Hence, malicious nodes can easily launch DoS or host

compromise attacks on the set of  $R$  replica holders of a target file ( $R \ll B$ ).

**Efficient Access Control.** A read-only file system like CFS can exercise access control by simply encrypting the contents of each file, and distributing the keys only to the legal users of that file. Farsite, a read-write file system, exercises access control using access control lists (ACL) that are maintained using a Byzantine-fault-tolerant protocol. However, access control is not truly distributed in Farsite because all users are authenticated by a small collection of directory-group servers. Further, PKI (public-key Infrastructure) based authentication and Byzantine fault tolerance based authorization are known to be more expensive than a simple and fast capability-based access control mechanism [21].

Bearing these issues in mind, in this chapter we present *LocationGuard* as an effective technique for countering targeted file attacks. The fundamental idea behind LocationGuard is to *hide* the very location of a file and its replicas such that, a legal user who possesses a file's *location key* can easily and securely locate the file on the overlay network; but without knowing the file's location key, an adversary would not be able to even locate the file, let alone access it or attempt to attack it. LocationGuard implements an efficient capability-based file access control mechanism through three essential components. The first component of LocationGuard is a location key, which is a random bit string (128 bits) used as a key to the location of a file in the overlay network, and addresses the capability revocation problem by periodic or conditional rekeying mechanisms. A file's location key is used to generate legal capabilities (tokens) that can be used to access its replicas. The second component is the routing guard, a secure algorithm to locate a file in the overlay network given its location key such that neither the key nor the location is revealed to an adversary. The third component is an extensible collection of location inference guards, which protect the system from traffic analysis based inference attacks, such as lookup frequency inference attacks, end-user IP-address inference attacks, file replica inference attacks, and file size inference attacks. LocationGuard presents a careful combination of location key, routing guard, and location inference guards, aiming at making it very hard for an adversary to infer the location of a target file by either actively or passively observing the overlay network.

In addition to providing an efficient file access control mechanism with traditional cryptographic guarantees like file confidentiality and integrity, LocationGuard mitigates Denial-of-Service (DoS) and host compromise attacks, while adding minimal performance overhead and small storage overhead to the file system. Our initial experiments quantify the overhead of employing LocationGuard and demonstrate its effectiveness against DoS attacks, host compromise attacks and various location inference attacks.

The rest of the chapter is organized as follows. Section 2.2 provides terminology and background on overlay network and serverless file systems like CFS and Farsite. Section 2.3 describes our threat model in detail. We present the core techniques of LocationGuard in Sections 2.6, 2.7, 2.8 and 2.9. We present a concrete implementation and a thorough experimental evaluation of LocationGuard in Section 2.10, related work in Section 2.11, and conclude the chapter in Section 2.12.

## ***2.2 Background and Terminology***

In this section, we formally describe a set of common properties of structured overlay networks. Our formal model brings out the important concepts behind DHT-based systems like Chord [106], CAN [77], Pastry [84] and Tapestry [10] that aid us in analyzing the vulnerabilities and security threats on structured overlay networks.

A typical DHT-based overlay network consists of a routing table based lookup service. The lookup service maps a given key to a node (usually the IP-address of the node) that is responsible for the key. Storage protocols are layered on top of the lookup protocol. For instance, CFS [23] is a wide-area cooperative file system layered on Chord [106]; while OceanStore [53] is a distributed file system layered on Tapestry [10]. A generic DHT-based lookup service has the following properties:

- **(P1) A key identifier space,  $K$ .**  $K$  is a  $m$ -bit identifier space where each data item is mapped to a unique identifier  $d \in K$  using any standard hash function (like MD5 [80] or SHA1 [32]).
- **(P2) ID Mapping Scheme** defines a node identifier space  $S$ . For example, Chord uses a one-dimensional circular identifier space; while CAN uses a  $d$ -dimensional

coordinate space. Each node  $n$  is assigned an identifier  $ID(n) \in S$ . Some DHT based systems (CAN [77]) allow nodes to choose their identifier, while most others derive the identifier of a node  $n$ , namely  $ID(n)$ , as a strong one-way function of an external identifier ( $EID$ ) of the node  $n$ . For example,  $ID(n)$  could be equal to  $hash(IP(n))$ , where  $IP(n)$  denotes the IP-address of node  $n$ . In this example, IP-address of a node is used as its external identifier.

- **(P3) Rules for dividing the node identifier space among the nodes.** The DHT-based schemes define a responsibility function for every node  $n$  which maps it to a contiguous segment of identifier space  $S$  at time  $t$ , denoted as  $Resp_t(n)$ . At any given time instant  $t$ ,  $\{Resp_t(n) \mid n \in N(t)\}$  partitions the node identifier space  $S$ , where  $N(t)$  refers to the total collection of all nodes in the system at time  $t$ . The algorithms also ensure that statistically every node  $n$  shares the identifier space equally; that is, at any time instant  $t$ ,  $sizeof(Resp_t(n)) \approx \frac{sizeof(S)}{N(t)}$ . Note that the function  $sizeof$  depends on the nature of the identifier space. For example, in Chord,  $sizeof(x)$  could be defined as the length of the segment  $x$ ; while in CAN,  $sizeof(x)$  could be defined as the *volume* of the coordinate space spanned by  $x$ .
- **(P4) Data Placement Scheme** specifies rules for mapping keys to nodes: A node  $n$  is responsible for a key  $k \in K$  at time  $t$  if and only if  $k \in Resp_t(n)$ . This guarantees that any key  $k$  would always be found since the set  $\{Resp_t(n) \mid n \in N(t)\}$  partitions the node identifier space  $S$ .
- **(P5) Routing Scheme** uses the per-node routing tables. Routing table entries on every node maintain references to other nodes. More specifically, a distance metric is defined between any two identifiers  $i$  and  $j$  as  $dist(i, j)$ . For example, in Chord,  $dist(i, j)$  may be simply defined as the length of the segment  $(i, j)$ ; while in CAN  $dist(i, j)$  could be defined as the Cartesian distance between the points  $i$  and  $j$  in a  $d$ -dimensional coordinate space. When a node  $n$  is queried for key  $k$ , it returns a node  $m$  that is *closer* to key  $k$ ; that is,  $dist(ID(n), k) \geq dist(ID(m), k)$ .
- **(P6) Rules for updating routing tables as nodes join and leave.** When a new

node  $m$  joins the network at time  $t$ , it typically contacts an existing node  $n \in N(t)$  such that  $ID(m) \in Resp_t(n)$ . Note that there always exists such a node  $n$  since the set  $\{Resp_t(n) \mid n \in N(t)\}$  partitions the identifier space  $S$ . The node  $m$  typically assumes responsibility over a portion of the identifier space mapped to node  $n$ ; that is;  $Resp_{t'}(n)$  and  $Resp_{t'}(m)$  partitions the space  $Resp_t(n)$  for  $t' > t$ . Similarly, when a node leaves the network, it hands over its responsibilities to another node in the system.

The DHT-based systems guarantee location of any data item within a bounded number of application level hops. However, this advantage comes with a price: the DHT-based systems enforce a highly rigid structure and rely heavily on the correct functioning of (almost) all nodes in the system. In short, an attacker can potentially harm the overlay network by targeting these delicately balanced structures enforced by DHT-based systems.

### ***2.3 Threat Model***

Adversary refers to a logical entity that controls and coordinates all actions by malicious nodes in the system. A node is said to be malicious if the node either intentionally or unintentionally fails to follow the system's protocols correctly. For example, a malicious node may corrupt the files assigned to them and incorrectly (maliciously) implement file read/write operations. This definition of adversary permits collusions among malicious nodes. We assume that the underlying IP-network layer may be insecure. However, we assume that the underlying IP-network infrastructure such as domain name service (DNS), and the network routers cannot be subverted by the adversary.

An adversary is capable of performing two types of attacks on the file system, namely, denial-of-service attacks, and host compromise attacks. When a node is under denial-of-service attack, the files stored at that node are unavailable. When a node is compromised, the files stored at that node could be either unavailable or corrupted. We model the malicious nodes as having a large but bounded amount of physical resources at their disposal. More specifically, we assume that a malicious node may be able to perform a denial-of-service attack only on a finite and bounded number of good nodes, denoted by  $\alpha$ . We limit

the rate at which malicious nodes may compromise good nodes and use  $\lambda$  to denote the mean rate per malicious node at which a good node can be compromised. For instance, when there are  $B$  malicious nodes in the system, the net rate at which good nodes are compromised is  $\lambda * B$  (*node compromises per unit time*). Every compromised node behaves maliciously. For instance, a compromised node may attempt to compromise other good nodes. Every good node that is compromised would independently recover at rate  $\mu$ . Note that the recovery of a compromised node is analogous to cleaning up a virus or a worm from an infected node. When the recovery process ends, the node stops behaving maliciously. Unless and otherwise specified we assume that the rates  $\lambda$  and  $\mu$  follow an exponential distribution.

## 2.4 Attacks on the Routing Scheme

A typical DHT-based overlay network constructs a topology in which every node plays the role of a client, a server, a router, and a domain name server. Nodes act as router cum domain name server when they translate an identifier to the IP-address of a node that is responsible for the identifier (see Property P4 in Section 2.2). Malicious nodes can potentially exploit this feature to misguide legitimate nodes with incorrect lookups. For example, a malicious node can lie about the next hop when it is queried for some identifier. This could result in denial of information - a legitimate node is denied access to a data item; or result in sub-optimal performance of the lookup algorithm.

There are several possible defense mechanisms to counteract such vulnerabilities. Concretely, the properties of the distributed lookup algorithm can be used to ascertain whether a lookup for a given identifier is correct or not. For example, Sit and Morris [93] exploit the fact that: at each hop of the Chord lookup algorithm the query originator knows that the lookup protocol should lead him/her *closer* to the destination identifier (see Property P5 in Section 2.2). Hence, the query originator can check for this and detect an incorrect lookup. On sensing an incorrect lookup, the query originator can choose an alternative (possibly sub-optimal) path towards the destination identifier. Informally, a *lookup path* from a source node  $n$  to a destination node  $m$  is the sequence of nodes through which the

lookup operation succeeds. In view of the above discussion, the performance of a lookup algorithm (in the presence of malicious nodes) depends on the following three factors: (i) Existence of multiple alternate paths between any two identifiers, (ii) Lookup costs along alternate paths between any two identifiers, and (iii) Ability to detect incorrect lookups.

### 2.4.1 Alternate Lookup Paths

We first highlight the importance of alternate (possibly sub-optimal) paths in enhancing the performance and the robustness of a lookup algorithm in the presence of malicious nodes. We capture the notion of *alternate lookup paths* using the notion of *independence* of lookup paths. We formally define independence between two lookup paths as follows:

**Definition** *Independent Lookup Paths:* Let  $P$  and  $Q$  be different lookup paths from node  $n$  to node  $m$ . Two lookup paths  $P$  and  $Q$  are said to be independent if and only if they do not share a common node other than the source node  $n$  and the destination node  $m$ .

Hence, each independent lookup path <sup>1</sup> between a node  $n$  and a node  $m$  is a *statistically independent* route for a lookup with key  $k \in Resp(m)$ , originated at node  $n$ , to succeed. Note that the property of independence is stronger than that of alternate paths. For instance, there may exist multiple paths between node  $n$  and node  $m$ ; however all these paths may happen to share a common node, thereby making no two of them independent.

Most of the DHT-based systems do not guarantee the existence of multiple independent lookup paths between any two identifiers. For instance, in Chord, all lookups for a key  $k \in Resp(m)$  will succeed only through the node  $pred(m)$ , where  $pred(m)$  denotes the predecessor of node  $m$  along the Chord identifier circle. Hence, the number of independent lookup paths between any node  $n$  and key  $k \in Resp(m)$  is one, since all such lookup paths include node  $pred(m)$ . If the node  $pred(m)$  were malicious, lookup for any key  $k \in Resp(m)$  would fail. On the other hand, this situation is greatly mitigated in DHT-based schemes like CAN that have multiple independent lookup paths between any two identifiers. More

---

<sup>1</sup>In general one can estimate number of independent paths as follows: By Menger's theorem [60], the number of independent paths equals the vertex connectivity of a graph; and vertex connectivity can be measured using network flow techniques [61]

specifically, a  $d$ -dimensional CAN topology has  $d$  independent lookup paths.

We use the *probability of lookup failure* as a metric for measuring the benefits of alternate lookup paths. A lookup for node  $m$  at node  $n$  results in a failure if *all* the lookup paths from node  $n$  to node  $m$  contain at least one malicious node. Intuitively, larger the number of independent lookup paths, smaller is the probability of lookup failure. In the following portions of this section we derive bounds on the probability of lookup failure in terms of the number of independent lookup paths.

#### 2.4.1.1 Quantitative Analysis

Let  $ind$  denotes the number of independent lookup paths between the source and destination node,  $p$  denotes the fraction of malicious nodes in the system, and  $M$  denotes the number of hops required for a lookup to succeed. Given  $ind$  independent lookup paths of length  $M$  hops, one can show that the probability of lookup failure is bounded by Equation 1.

$$p^{ind} \leq Pr(\text{lookup failure}) \leq (1 - (1 - p)^M)^{ind} \quad (1)$$

Note that the existence of  $ind$  independent paths between a source node  $src$  and destination node  $dst$  implies that there exists nodes  $\{n_1, n_2, \dots, n_{ind}\}$  one of which occurs on all paths from the node  $src$  to node  $dst$ . The lower bound is derived from the fact that a lookup from node  $src$  for node  $dst$  is guaranteed to fail if all the  $ind$  nodes  $\{n_1, n_2, \dots, n_{ind}\}$  were malicious. Let  $\{P_1, P_2, \dots, P_{ind}\}$  be any set of  $ind$  independent lookup paths between node  $src$  and node  $dst$  containing nodes  $\{n_1, n_2, \dots, n_{ind}\}$  respectively. The probability of a lookup succeeding on any lookup path  $P_i$  with  $M$  hops equals  $(1 - p)^M$ , namely, the probability that all the nodes on that path were good. The upper bound follows from the independence of lookup failures along each independent lookup path <sup>2</sup>. For small values of  $p$ , the probability of lookup failure can be approximated to  $(M * p)^{ind}$  ( $M * p \ll 1$ ). Intuitively, the longer a lookup path ( $M$ ), the higher is the chance that at least one node on the lookup path turns out malicious. The statistical independence in lookup failures along multiple independent paths ensures that the probability of lookup failure decreases

---

<sup>2</sup>This is an upper bound because the presence of alternate (but not independent) lookup paths may decrease the probability of lookup failure

*exponentially* with the number of independent paths (*ind*).

### 2.4.2 Alternate Optimal (less costly) Lookup Paths

Yet another important issue to be addressed with regard to alternate lookup paths is the cost of these alternative paths themselves. Ideally, the alternate paths should be *alternate optimal* paths; otherwise, choosing highly sub-optimal alternate paths may degrade the performance of a lookup algorithm. Unfortunately, most of the DHT-based schemes do not address such issues. For illustration, in Chord, say a node  $n$  queries a good node  $x$  for key  $k$  and obtains the result as node  $y$ . Now, node  $n$  issues a query for key  $k$  to node  $y$ . If node  $y$  were malicious, it would return an incorrect lookup result. If node  $n$  were to detect the invalid result, the best choice it has is to ask node  $x$  (previous node on the lookup path) for its *next best choice* for the query with key  $k$ . Now, node  $x$  has to return a sub-optimal result, since it is *not aware* of any node that is closer to key  $k$  than the malicious node  $y$ . Since Chord maintains pointers to nodes at distance that are an integer power of 2, it is likely that the next best choice proceeds only half the distance along the identifier circle when compared to the optimal choice. On the other hand, in CAN it is quite possible for the alternate paths to be *near optimal*. Consider the same scenario described above. Assume, without loss of generality, that the identifier of node  $x$  and key  $k$  differ along coordinates  $\{c_1, c_2, \dots, c_l\}$ . Now, if node  $x$  and node  $y$  varied along a coordinate  $c_j$  (for some  $j$ ,  $1 \leq j \leq l$ ), then node  $x$  could choose other neighboring nodes that vary along coordinates other than  $c_j$ . In comparison with Chord, the alternate choices provided for a lookup in CAN, is likely to be much closer to optimality. We defer further discussion on alternate-optimal paths to the end of this section.

### 2.4.3 Detecting and Recovering from Invalid Lookups

Having highlighted the importance of good alternate paths, we now study the importance of detecting incorrect (malicious) lookups. In the discussion that follows, we assume two failure modes for nodes in our system: *Crash failures* and *Byzantine failures*. When a node has crash failed it does not return any results for lookup queries. Under Byzantine failure, a node can return a potentially malicious value for any lookup query. In the following

portions of this section, we quantitatively analyze the cost of lookup operation under both these failure modes. For the sake of simplicity of analysis, we assume that the DHT-based scheme has multiple alternate-optimal paths between any two identifiers (like CAN). Hence, the results obtained from the results of our analysis below can be viewed as lower bounds on the lookup costs. Algorithm 1 shows the algorithm used by legitimate nodes to detect and recover from invalid lookup results.

```

SECURE_LOOKUP(key  $k$ )
(1) for all good nodes  $n$ 
(2)    $m \leftarrow n$ 
(3) repeat
(4)    $m \leftarrow \text{lookup}(m, k)$ 
(5)   if  $k \notin \text{Resp}(m)$  AND  $\text{dist}(m, k) > \text{dist}(\text{prev}(m), k)$ 
(6)     repeat
(7)        $m \leftarrow$  next best node for key  $k$  from node  $\text{prev}(m)$ 
(8)       if  $m = \text{nil}$ 
(9)          $m \leftarrow \text{prev}(m)$ 
(10)      end if
(11)      if  $m = \text{nil}$ 
(12)        Report lookup failed
(13)      end if
(14)      until  $m \neq \text{nil}$ 
(15)    end if
(16)  until  $k \in \text{Resp}(m)$ 
(17) end for

```

**Figure 1:** Lookup algorithm for legitimate nodes

### 2.4.3.1 Quantitative analysis

Let  $M$  denote the mean number of hops required to perform a lookup operation. For instance, in Chord,  $M = \frac{1}{2} \log(N)$ ; in CAN,  $M = \frac{d}{4} N^{\frac{1}{d}}$  where  $N$  is the number of nodes in the system and  $d$  represents the dimensionality of CAN's coordinate space. Let  $p$  denote the percentage of bad nodes in the system. Also assume that the bad nodes are uniformly spread throughout the node identifier space. Let  $f(x)$  be a function that maps the number of hops required for a lookup when all nodes are good to the number of hops required when  $p\%$  of the nodes are malicious. In other words, if a lookup would require  $x$  hops when all nodes are good, it would require  $f(x)$  hops when  $p\%$  of the nodes are bad.

**Crash Failures.** Assuming crash failures for nodes,  $f_1(x)$  (the mapping function for crash failures) satisfies the following recurrence relation.

$$f_1(x) = 1 + (1 - p)f_1(x - 1) + pf_1(x) \quad (2)$$

When a node  $n$  queries a node  $m$  for the next hop towards a key identifier  $k$ , node  $n$  expends one hop. If node  $m$  is a good node (probability =  $1 - p$ ) then it would return a correct lookup result. Hence, node  $n$  would have to traverse  $x - 1$  more hops to reach the key identifier  $k$  in the scenario where no node has crashed. In the presence of crash-failed nodes, this would require  $f_1(x - 1)$  hops by the definition of function  $f_1$ . If node  $m$  had crashed (probability =  $p$ ), it would not return any lookup result. Node  $n$  on detecting this (though a timeout mechanism) can choose an alternate-optimal path towards key  $k$ . Hence, node  $n$  would have to traverse  $x$  more hops to reach the key identifier  $k$  in the scenario where no node has failed. In the presence of crash-failed nodes, this would cost node  $n$  additional  $f_1(x)$  hops (it is still possible to reach key  $k$  in  $x$  hops in spite of ruling out one lookup path, since we have assumed the presence of alternate-optimal pathss).

Solving the recurrence relation, we get,

$$f_1(x) = \frac{x}{1 - p} \quad (3)$$

Hence, the expected (average) number of hops required for a lookup operation is,  $E[f_1(x)] = \frac{M}{1-p}$  since,  $M = E[x]$  by definition.

**Byzantine Failures.** Assuming Byzantine failure of nodes, the cost of a lookup operation depends on certain properties of the DHT-based system. In most of the DHT-based systems it is possible to detect invalid lookups with a reasonable degree of certainty since the lookup at each hop is *supposed* to get closer to the destination identifier [93]. Hence, the query originator can check for this and detect an incorrect lookup. Upon finding an incorrect lookup, the query originator can choose an alternative (possibly sub-optimal) path towards the destination identifier. However, in certain cases like CAN's RTT optimization, the

lookup results cannot be verified since the intermediate lookup results are not available to the source node (query originator). In view of the above discussion, we consider the following two scenarios:

- Scenario 1: An incorrect lookup can always be detected.
- Scenario 2: An incorrect lookup cannot be detected; and hence, the querier blindly follows the lookup result.

Observe that *Scenario 1* is simply equivalent to that of crash failure of nodes. Note that if a non-malicious node  $n$  receives an incorrect lookup from a malicious node  $m$  then node  $n$  can simply assume that node  $m$  has crash failed. Hence, the lookup cost in scenario 1 is given by Equation 3.

Assuming that incorrect lookups can neither be detected or corrected,  $f_2(x)$  (the mapping function for scenario 2) satisfies the following recurrence relation,

$$f_2(x) = 1 + (1 - p)f_2(x - 1) + pf_2(M) \quad (4)$$

Note that the first two terms in the expression for  $f_2$  follows from the same arguments for crash failures; it costs unity for node  $n$  to query  $m$  and  $f_2(x - 1)$  additional hops if node  $m$  were good. However, if node  $m$  were malicious, it would return an incorrect lookup and node  $n$  would blindly abide by node  $m$ 's result. Note that a collection of malicious nodes  $X$  may keep circulating the query among nodes in  $X$ , thereby ensuring that the query never succeeds. However, this would cost bad nodes in terms of their bandwidth for answering repeated queries. Hence, we assume that bad nodes return a random node in the system as the next hop for key  $k$  to the query originator node  $n$ . Now, since the random node could be located anywhere in the network, it would be  $M$  hops away from the key  $k$  in the scenario where all nodes are good. Hence, in the presence of malicious nodes, the lookup operation would cost node  $n$  additional  $f_2(M)$  hops.

Using the recurrence relation 4 we compute the average number of hops required for a lookup operation as follows. We approximate  $E[f_2(x)]$  to  $f_2(E[x])$  which is equal to  $f_2(M)$  (since  $M = E[x]$ ). Note that  $f_2(M)$  denotes the lookup cost for scenario 2 in the

presence of malicious nodes when it would have required  $M$  hops in the absence of malicious nodes. We observed that in most DHT-based systems the mean number of hops  $M$  is also the most probable number of hops between any two random nodes in the system. Hence, such an approximation does not significantly perturb our analytical results. Further, our experimental results in Figure 4 show that this approximation is acceptable. Hence,

$$E[f_2(x)] \approx f_2(M) = \frac{1 - (1 - p)^M}{p(1 - p)^M} \quad (5)$$

**Summary.** Clearly, scenario 2 pays higher penalty for its inability to detect and recover from invalid lookups. Intuitively, in scenario 1 (or under crash failures), the lookup makes one successful hop with a probability  $1 - p$ ; hence, each hops translates into  $\frac{1}{1-p}$  hops. On the other hand, scenario 2, pays heavily for every failed lookup. Let us say that we start with a state  $S$  where the lookup operation is  $M$  hops from its target. Now, this lookup operation succeeds if all the nodes in the path to the target are good ( $Pr = (1 - p)^M$ ); else it is back to its original state  $S$ . Hence, the probability that a lookup succeeds in any given attempt starting from state  $S$  is  $(1 - p)^M$ ; and hence the lookup cost is varies as  $(1 - p)^{-M}$ .

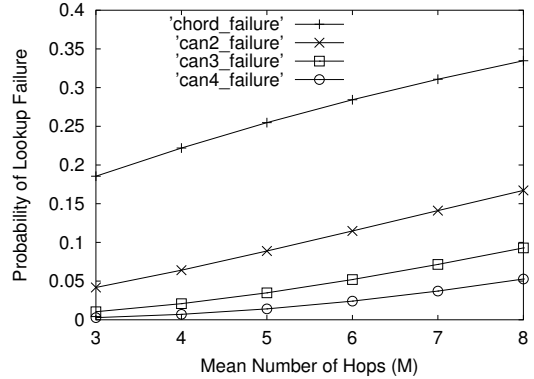
#### 2.4.4 Experimental Validation

We have so far identified and quantitatively analyzed the importance of multiple independent paths, alternate optimal paths, the ability to detect and recover from invalid lookups. Now we present two sets of experiments to validate the above analysis. First, we study the dependency between the number of independent paths and the probability of lookup failure. Second, we measure the lookup cost in the presence of malicious nodes and evaluate the performance of the proposed defense mechanisms regarding to the two scenarios: An incorrect lookup (1) can always be detected or (2) cannot be detected.

**Experiment I.** In this experiment, we demonstrate that having multiple independent lookup paths indeed decreases the probability of lookup failures. We simulated the working of a P2P system using the Chord lookup protocol with 1024 nodes. The average lookup cost when there are no malicious nodes is 5, i.e.,  $M = 5$ . We also constructed CAN systems with approximately the same average lookup cost; a 2-dimensional CAN with 100 nodes

DHT	Lower Bound	Expt Result	Upper Bound
Chord	0.1	0.29	0.40
CAN-2	0.01	0.09	0.16
CAN-3	0.001	0.04	0.06
CAN-4	0.0001	0.015	0.028
CAN-10	0.0	$10^{-4}$	$10^{-4}$

**Table 1:** Probability of Lookup Failure ( $p = 10\%$ ): Quantitative Analysis



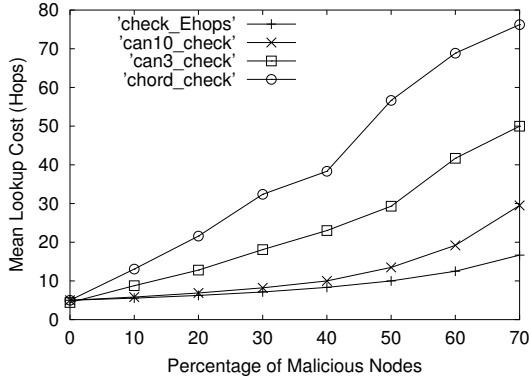
**Figure 2:** Probability of a lookup failure: Initial  $TTL = 100$  and  $p = 10\%$

( $M = 5$ ), a 3-dimensional CAN with 216 nodes ( $M = 4.5$ ), a 4-dimensional CAN with 625 nodes ( $M = 5$ ), and a 10-dimensional CAN with 1024 nodes ( $M = 5$ ). A random set of  $p\%$  of the nodes were chosen to behave maliciously. From a practical standpoint, we associate a time-to-live ( $TTL$ ) with every lookup operation. Hence, a lookup operation is successful only if it terminates correctly within  $TTL$  overlay network hops.

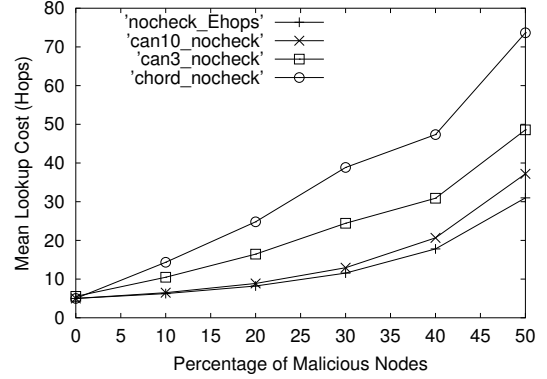
Table 1 compares the experimental results with the bounds obtained from our analytical model (Equation 1, Section 2.4.1) when  $p = 10\%$ . Although the bounds obtained from our quantitative analysis are not absolutely tight (because Equation 1 does not consider alternate but not independent paths), the trends revealed by our analysis closely reflect the results obtained from our experiments. Most importantly, observe that the upper bound on the probability of lookup failure sharply decreases with increase in the number of independent lookup paths. This motivated us to experiment with overlay network with different number of independent paths.

Figure 2 shows the probability of lookup failure with  $TTL = 100$  and  $p = 10\%$  and varying  $M$ . Observe that the probability of lookup failure increases with the mean number of hops ( $M$ ). Note that for any lookup path to succeed, all the nodes on the lookup path must be non-malicious; longer the path, higher is the probability that at least one malicious node appears on that path.

**Experiment II.** In this experiment, we illustrate the performance of DHT-based systems



**Figure 3:** Lookup Costs: Scenario 1 with  $M = 5$



**Figure 4:** Lookup Cost: Scenario 2 with  $M = 5$

for the scenarios 1 and 2 discussed in section 2.4.3. Figure 3 shows the average lookup cost for scenario 1 wherein, the legitimate nodes verify whether the lookup result *appears* to be valid by checking if the new node is indeed closer to the key  $k$ . On detecting an invalid lookup, node  $n$  gets the next best alternative node and forwards the query to it. Note that this strategy for detecting invalid lookups has scope for an *one-sided error*; it may conclude that an incorrect (or a sub-optimal) lookup result is correct. The estimates from our quantitative analysis are labeled as ‘check\_Ehops’ (Equation 3, Section 2.4.3); they denote the lower bounds obtained on the expected number of hops assuming a large number of independent paths. The lines labeled ‘chord\_check’ and ‘cand\_check’ refer to the cases wherein the lookup protocol checks for validity as a part of the Chord, and the  $d$ -dimensional CAN protocols respectively. We shall discuss the implications of Figure 3 in conjunction with Figure 4. Figure 4 shows the average lookup cost for scenario 2 wherein, the legitimate nodes do not test the validity of lookup results. The line labeled ‘nocheck\_Ehops’ (Equation 5, Section 2.4.3) shows the results obtained from our quantitative analysis. The lines labeled ‘chord\_nocheck’ and ‘cand\_nocheck’ refer to the cases where the Chord and the  $d$ -dimensional CAN protocols were used without checking for the validity of lookup operations. Based on Figures 3 and 4, we can testify the following statements:

- *Validate our quantitative analysis.* Observe that ‘check\_Ehops’ and ‘nocheck\_Ehops’ act as lower bounds for the ‘check’ and the ‘nocheck’ versions of the lookup protocol

respectively. Also, observe that the results for a 10-D CAN closely matches our quantitative analysis, since our analysis was specifically targeted at obtaining lower bounds on lookup costs assuming a large number of independent paths between any two identifiers.

- *Checking the validity of a lookup result* becomes very important for large values of  $p$ . However, for small values of  $p$ , checking the validity of a lookup result is not very vital in the presence of multiple independent paths. In fact, for  $p \leq 40\%$ , a 10-D CAN with no validity checks incurs no more than 5-8% higher lookup cost than a 10-D CAN that performs validity checks. But, for large values of  $p$  (around  $p = 70\%$ ), lookup protocols that do not include validity checks incur as high as 40-50% (for 10-D CAN) to about 200% (for Chord) (Note that Figure 4 shows the results only up to  $p = 50\%$ ).
- *Importance of good alternate paths.* Since ‘can10\_check’ has multiple near-optimal alternate paths, its lookup cost is within twice of the optimal lookup cost even when  $p$  equals 70%. On the other hand, ‘chord\_check’ shows much poorer performance, primarily because of the fact that Chord does not provide multiple independent lookup paths. Further, the vitality of alternate paths is more blatantly revealed by Figure 4 from the fact that, ‘chord\_check’ performs much worse (3-4 times) than ‘can10\_nocheck’.

## 2.5 Attacking the ID-to-Key Mapping Scheme

In this section, we present the ID-to-Key mapping attack (ID Mapping for short) that show how the malicious nodes could exploit the identifier-to-key mapping to corrupt a *chosen* data item stored in the system by spoofing multiple identities (*pseudo-spoofing*). We quantify and analyze the cost of attacking a chosen data item and discuss some approaches to mitigate this problem.

The famous Sybil Attack chapter [30] showed that entities (nodes) can forge multiple identities for malicious intent, and hence, a small set of faulty entities may be represented

through a larger set of identities. Douceur concludes in [30] that for direct or indirect identity validation (without using a centralized trusted agency), a set of faulty nodes can counterfeit an unbounded number of identities (pseudo-spoofing). One solution suggested to counter the Sybil attack in [30] is using secure node IDs that are signed by well-known trusted certifying authorities. However, as Douceur pointed out himself that mandating that every node must possess a certificate would turn out expensive. Hence, one is forced to employ weak secure node IDs (with challenge-response schemes to verify the node IDs); for example, several systems like CFS [23] use the IP-address of a node as its weak secure ID. Therefore, it becomes very important to quantify and analyze the security trade-offs when weak secure IDs are used. This section discusses an extension of the pseudo-spoofing attack which results in the loss of a chosen data item ( $d$ ) assuming the identity of the data item ( $ID(d)$ ) is known.

Almost all DHT-based system use a strong one-way hash function (like MD5 [80] or SHA1 [32]) to derive a node identifier from its external identifier (EID) (refer Section 2.3); and, any node  $p$  has direct access to a data item  $d$  if and only if  $ID(d) \in Resp(p)$  (see properties P2 and P4, Section 2.2). Therefore, for a malicious node  $n$  to target a data item with identifier  $d$ , it needs to select an EID such that if node  $n$  joins the system with  $ID(n) = hash(n.EID)$ , it will be made responsible for the target data item  $d$ . However, we show that the cost of attacking a specific data item  $d \in K$  (key identifier space  $K$ ) is much *easier* than inverting the ID mapping hash function. Recall that by birthday paradox [117], the cost of inverting a strong one-way hash function is  $O(\sqrt{sizeof(S)})$ , where  $S$  is the hash space (in our case, the identifier space). In this section, we present a  $O(G)$  attack for attacking any chosen data item stored in the system. This attack assumes significance because the number of good nodes  $G$  is of the order of a few thousands, while  $sizeof(S)$  for say MD5 is  $2^{128}$ .

Concretely, given the identifier of a data item  $d$  a malicious node can locate the node at which the data item  $d$  is stored using the lookup protocol. Let the data item  $d$  be stored at node  $q$  at time  $t$ , namely,  $ID(d) \in Resp_t(q)$ . A malicious node  $p$  gains access to the target data item  $d$  as follows: First, it attempts to pick an EID that hashes into  $Resp_t(q)$ . As we

have described in section 2.2 (P6), when a node  $p$  with  $ID(p) \in Resp_t(q)$  joins the network, it would share the responsibility of node  $q$ . Given that the node  $p$  gets assigned a portion of the node  $q$ 's responsibility, there is a good chance that the target data item  $d$  indeed gets assigned to node  $p$ . Observe that if the system did not enforce restrictions on a node's identifier, a malicious node  $p$  could trivially choose its node identifier to be  $ID(d)$  thereby ensuring that the target data item  $d$  is assigned to it. Algorithm 5 shows the algorithm used by malicious nodes to perform a target data item attack.

```

ID_MAPPING_ATTACK(data_item  $d$ )
(1)  $m \leftarrow Lookup_p(d)$ 
(2) repeat
(3)    $EID(n) =$  Set of EIDs owned by node  $n$ 
(4)    $eid \leftarrow$  Randomly choose an EID address from set  $EID(n)$ 
(5)    $ID(n) \leftarrow hash(eid)$ 
(6) until  $ID(n) \in Resp(m)$ 
(7) Join the system with identifier as  $ID(n)$ 
(8) if data item  $d$  is assigned to node  $n$ 
(9)   Corrupt data item  $d$ 
(10) else
(11)   Repeat procedure ID_Mapping_Attack
(12) end if

```

**Figure 5:** ID Mapping scheme: attack a specific data item  $d$

### 2.5.1 Quantitative Analysis

Let  $G$  denote the number of good nodes in the system. Assume for the sake of simplicity that there are no malicious nodes in the system. Now, every node in the system would be responsible for approximately  $1/G^{th}$  portion of the identifier space. The identifier space can be viewed as if it were divided into  $G$  equal sized buckets. Hence, the probability that a random identifier falls into a given bucket  $q$  is  $1/G$ . The probability that a malicious node hits upon an EID that hashes into bucket  $q$  in its  $k^{th}$  attempt is given by,

$$Pr(k \text{ attempts}) = \left(1 - \frac{1}{G}\right)^{k-1} \frac{1}{G} \quad (6)$$

where the first  $k - 1$  attempts fails to fall into bucket  $q$ , while the  $k^{th}$  attempt succeeds. Observe that the number of attempts required in Equation 6 follows a Geometric Distribution.

The malicious nodes could choose  $q$  such that their target data item  $d$  is currently held by node  $q$ , i.e.,  $ID(d) \in Resp_t(q)$ . Using the standard properties of a Geometric distribution, it follows from Equation 6 that the expected number of attempts required to obtain an identifier that hashes into node  $q$  is  $G$ . Also, the probability that more than  $G$  attempts are required is  $\frac{1}{e}$  for substantially large values of  $G$ . But, having obtained an identifier that hashes into node  $q$ 's identifier space does not guarantee that the malicious node  $p$  is assigned the target data item  $d$ . (Note  $Resp_{t'}(p)$  and  $Resp_{t'}(q)$  partitions  $Resp_t(q)$ , for  $t' > t$  (see property P6 in Section 2.2). Since the data item  $d$  can lie anywhere in the identifier space assigned to node  $q$ , the probability that node  $p$  gets to store data item  $d$  after it joins the network is  $\frac{1}{2}$ . Hence, with a reasonably large probability, a malicious node  $p$  can obtain access to a target data item  $d$  in  $G$  attempts. Hence, an adversary that possesses  $G$  pseudo-identifiers can obtain access to the target file replica  $f_i$  with probability  $Pr(N_{EID} \leq G) = 1 - \frac{1}{2} \left(1 - \frac{1}{e}\right)$  (equivalently,  $Pr(N_{EID} > G) = 1 - \frac{1}{2} \left(1 - \frac{1}{e}\right) = \frac{1}{2} \left(1 + \frac{1}{e}\right)$ ). Consequently, one can improve the chances of this attack in  $O(G)$  attempts since, for some integer  $c > 0$ :

$$Pr(\text{Number of attempts} > cG) = \left(\frac{1}{2} \left(1 + \frac{1}{e}\right)\right)^c \quad (7)$$

In a system where  $R$  replicas are maintained for each data item, a group of  $B$  malicious nodes may join hands to corrupt a data item  $d$  irrecoverably as follows. Each of the malicious nodes performs an ID Mapping attack using the above strategy on any of the  $R$  replicas of data item  $d$ . When the malicious nodes succeed in gaining control over  $cr$  copies of the data item  $d$ , they can corrupt it and leave the system.

Consider the case wherein the IP-address of a node is used as its EID. Given the fact that IPv6 can potentially provide every node with thousands of addresses, it is quite feasible, though expensive, for a malicious node to perform this attack. The same argument is also applicable to dial-up users who obtain Dynamic IP-address from a huge ISP (Internet Service Provider). Also note that computing  $O(G)$  hashes is computationally feasible. As a simple example, using the standard OpenSSL library, a typical 900MHz Pentium III takes about 1 second to compute one million hashes (MD5).

$G$	Mean time (ms)	$E[N_{EID}]$	$\Pr(N_{EID} \leq 4G)$	$\Pr(N_{EID} \leq 8G)$
1024	2.32	2112	0.80	0.96
2048	4.22	4301	0.76	0.95
4096	8.47	8157	0.76	0.97
8192	16.08	16421	0.84	0.99

**Table 2:** Attack on ID Mapping Scheme

### 2.5.2 Experimental Validation

We simulated the Chord lookup protocol [106] with varying number of good nodes. We chose 100 random data items to attack. Table 2 summarizes our observations on the number of EIDs required for a successful attack on these data items. Our observations very closely match the results from our analysis (Equation 7):  $\Pr(N_{EID} \leq 4G) = 0.79$  and  $\Pr(N_{EID} \leq 8G) = 0.96$ .

### 2.5.3 Defense

The Sybil attack chapter [30] suggests the use of trusted certification authorities to strictly bind an identity to an entity (node). However this requires every node to reveal its identity in order to join the system. It may also discourage a few nodes from joining the overlay network in the first place. In order to reduce certification costs and motivate users to join the overlay network, one could employ weak secure identifiers, like the node IP-address, which can be challenged and verified easily.

There are other approaches that could mitigate this problem. When a new node joins the system, it typically contacts a publicly known and trusted bootstrap server to obtain an *entry point* node to bootstrap itself into the system. In our solution, the bootstrap server assigns a random  $id \in S$  to a node (and issues a certificate with a short life-time) when it joins the system. Observe that if a malicious node joins the system  $O(G)$  times, it gets assigned a given target data item with a large probability as in Equation 7. However note that contrary to the techniques that derive a node’s ID from its EID, a malicious node cannot *offline* determine the EID that can be used to gain control over the target data item. Instead, the malicious node has to physically attempt joining the system  $O(G)$  times, each

time contacting the bootstrap server. Note that contacting the bootstrap server  $O(G)$  takes significantly longer time; but, it does not prevent a malicious from eventually gaining control over a target data item. To prevent this, one could implement weak security checks through the bootstrap server; for example, the bootstrap server could detect frequent attempts by a node from a single *IP-domain*<sup>3</sup> to join the system. In conclusion, the overlay network may use weak secure IDs as long as spoofing a large number of pseudo-identifiers over a short duration of time is very hard.

## 2.6 *LocationGuard*

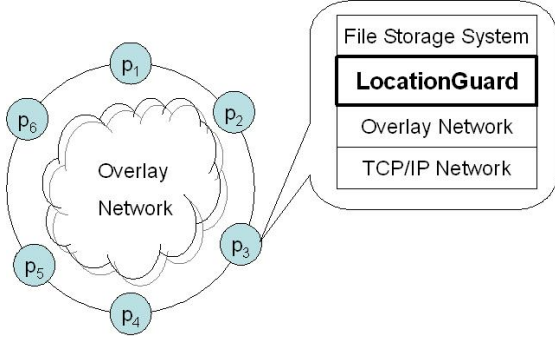
### 2.6.1 Targeted File Attacks

Targeted file attack refers to an attack wherein an adversary attempts to attack a small (chosen) set of files in the system. An attack on a file is successful if the target file is either rendered unavailable or corrupted. Given  $R$  replicas of a file  $f$ , file  $f$  is unavailable (or corrupted) if at least a threshold  $cr$  number of its replicas are unavailable (or corrupted). For example, for read/write files maintained by a Byzantine quorum [7],  $cr = \lceil R/3 \rceil$ . For encrypted and authenticated files,  $cr = R$ , since the file can be successfully recovered as long as at least one of its replicas is available (and uncorrupt) [23]. Most P2P trust management systems such as [119] uses a simple majority vote on the replicas to compute the actual trust values of peers, thus we have  $cr = \lceil R/2 \rceil$ .

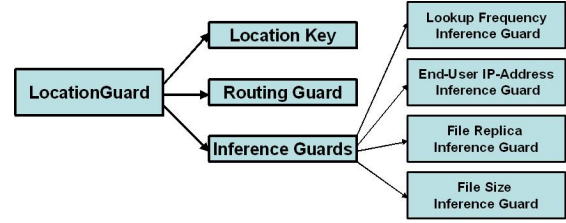
Distributed file systems like CFS and Farsite are highly vulnerable to target file attacks since the target file can be rendered unavailable (or corrupted) by attacking a *very small* set of nodes in the system. The key problem arises from the fact that these systems store the replicas of a file  $f$  at *publicly known* locations [51] for easy lookup. For instance, CFS stores a file  $f$  at locations derivable from the public-key of its owner. An adversary can attack any set of  $cr$  replica holders of file  $f$ , to render file  $f$  unavailable (or corrupted). Farsite utilizes a small collection of publicly known nodes for implementing a Byzantine fault-tolerant directory service. On compromising the directory service, an adversary could obtain the locations of all the replicas of a target file.

---

<sup>3</sup>Multiple IP-addresses owned by a node hopefully fall into the same IP-domain



**Figure 6:** LocationGuard: System Architecture



**Figure 7:** LocationGuard: Conceptual Design

Files on an overlay network have two primary attributes: (i) *content* and (ii) *location*. File content could be protected from an adversary using cryptographic techniques. However, if the location of a file on the overlay network is publicly known, then the file holder is susceptible to DoS and host compromise attacks. LocationGuard provides mechanisms to hide files in an overlay network such that only a legal user who possesses a file’s location key can easily locate it. Thus, any previously known attacks on file contents would not be applicable unless the adversary succeeds in locating the file. It is important to note that LocationGuard is oblivious to whether or not file contents are encrypted. Hence, LocationGuard can be used to protect files whose contents cannot be encrypted, say, to permit arbitrary regular expression based keyword search on the file contents.

### 2.6.2 LocationGuard Approach

We first present a high level overview of LocationGuard. Figure 6 shows an architectural overview of a file system powered by LocationGuard. LocationGuard operates on top of an overlay network of  $N$  nodes. Figure 7 provides a sketch of the conceptual design of LocationGuard. LocationGuard scheme guards the location of each file and its access with two objectives: (1) to hide the actual location of a file and its replicas such that only legal users who hold the file’s location key can easily locate the file on the overlay network, and (2) to guard lookups on the overlay network from being eavesdropped by an adversary. LocationGuard consists of three core components. The first component is *location key*, which controls the transformation of a filename into its location on the overlay network,

analogous to a traditional *cryptographic key* that controls the transformation of plaintext into ciphertext. The second component is the *routing guard*, which makes the location of a file unintelligible. The routing guard is, to some extent, analogous to a traditional *cryptographic algorithm* which makes a file’s contents unintelligible. The third component of LocationGuard includes an extensible package of location inference guards that protect the file system from indirect attacks. Indirect attacks are those attacks that exploit a file’s metadata information such as file access frequency, end-user IP-address, equivalence of file replica contents and file size to infer the location of a target file on the overlay network.

In the following subsections, we first present the main concepts behind location keys and location hiding (Section 2.6.3) and describe a reference model for serverless file systems that operate on LocationGuard (Section 2.6.4). Then we present the concrete design of LocationGuard’s three core components: the location key (Section 2.7), the routing guard (Section 2.8) and a suite of location inference guards (Section 2.9).

### 2.6.3 Concepts and Definitions

In this section we define the concept of location keys and its location hiding properties. We discuss the concrete design of location key implementation and how location keys and location guards protect a file system from targeted file attacks in the subsequent sections.

Consider an overlay network of size  $N$  with a Chord-like lookup protocol  $\Gamma$ . Let  $f^1, f^2, \dots, f^R$  denote the  $R$  replicas of a file  $f$ . Location of a replica  $f^i$  refers to the IP-address of the node (replica holder) that stores replica  $f^i$ . A file lookup algorithm is defined as a function that accepts  $f^i$  and outputs its location on the overlay network. Formally we have  $\Gamma : f^i \rightarrow loc$  maps a replica  $f^i$  to its location  $loc$  on the overlay network  $P$ .

**Definition 1** *Location Key*: A location key  $lk$  of a file  $f$  is a relatively small amount ( $m$ -bit binary string, typically  $m = 128$ ) of information that is used by a Lookup algorithm  $\Psi : (f, lk) \rightarrow loc$  to customize the transformation of a file into its location such that the following three properties are satisfied:

1. Given the location key of a file  $f$ , it is *easy* to locate the  $R$  replicas of file  $f$ .

2. Without knowing the location key of a file  $f$ , it is *hard* for an adversary to locate any of its replicas.
3. The location key  $lk$  of a file  $f$  should not be exposed to an adversary when it is used to access the file  $f$ .

Informally, location keys are *keys with location hiding property*. Each file in the system is associated with a location key that is kept secret by the users of that file. A location key for the file  $f$  determines the locations of its replicas in the overlay network. Note that the lookup algorithm  $\Psi$  is publicly known; only a file’s location key is kept secret.

Property 1 ensures that valid users of a file  $f$  can easily access it provided they know its location key  $lk$ . Property 2 guarantees that illegal users who do not have the correct location key will not be able to locate the file on the overlay network, making it harder for an adversary to launch a targeted file attack. Property 3 warrants that no information about the location key  $lk$  of a file  $f$  is revealed to an adversary when executing the lookup algorithm  $\Psi$ .

Having defined the concept of location key, we present a reference model for a file system that operates on LocationGuard. We use this reference model to present a concrete design of LocationGuard’s three core components: the location key, the routing guard and the location inference guards.

#### 2.6.4 LocationGuard File System

A serverless file system may implement read/write operations by exercising access control in a number of ways. For example, Farsite [7] uses an access control list maintained among a small number of directory servers through a Byzantine fault tolerant protocol. CFS [23], a read-only file system, may implement access control by encrypting the files and distributing the file encryption keys only to the legal users of a file. In this section we show how a LocationGuard based file system exercises access control.

In contrast to other serverless file systems, a LocationGuard based file system does not directly authenticate an user attempting to access a file. Instead, it uses location keys to implement a capability-based access control mechanism, that is, any user who presents the

correct file capability (token) is permitted access to that file. Furthermore, it utilizes routing guard and location inference guards to secure the locations of files being accessed on the overlay network. Our access control policy is simple: *if you can name a file, then you can access it*. However, we do not use a file name directly; instead, we use a pseudo-filename (128-bit binary string) generated from a file’s name and its location key (see Section 2.7 for detail). The responsibility of access control is divided among the file owner, the legal file users, and the file replica holders and is managed in a decentralized manner.

**File Owner.** Given a file  $f$ , its owner  $u$  is responsible for securely distributing  $f$ ’s location key  $lk$  (only) to those users who are authorized to access the file  $f$ .

**Legal User.** A user  $u$  who has obtained the valid location key of file  $f$  is called a legal user of  $f$ . Legal users are authorized to access any replica of file  $f$ . Given a file  $f$ ’s location key  $lk$ , a legal user  $u$  can generate the replica location token  $rlt^i$  for its  $i^{th}$  replica. Note that we use  $rlt^i$  as both the pseudo-filename and the capability of  $f^i$ . The user  $u$  now uses the lookup algorithm  $\Psi$  to obtain the IP-address of node  $r = \Psi_{lk}(rlt^i)$ . User  $u$  gains access to replica  $f^i$  by presenting the token  $rlt^i$  to node  $r$ . Note that  $rlt^i$  acts as a pseudo-filename during lookup and a capability during access control.

**Good Replica Holder.** Assume that a node  $r$  is responsible for storing replica  $f^i$ . Internally, node  $r$  stores this file content under its pseudo-filename  $rlt^i$ . Note that node  $r$  does not need to know the actual file name ( $f$ ) of a locally stored file  $rlt^i$ . Also, by design, given the internal file name  $rlt^i$ , node  $r$  cannot guess its actual file name (see Section 2.7). When a node  $r$  receives a read/write request on a file  $rlt^i$  it checks if a file named  $rlt^i$  is present locally. If so, it *directly* performs the requested operation on the local file  $rlt^i$ . Access control follows from the fact that it is very hard for an adversary to guess correct file tokens.

**Malicious Replica Holder.** Let us consider the case where the node  $r$  that stores a replica  $f^i$  is malicious. Note that node  $r$ ’s response to a file read/write request can be undefined. Note that we have assumed that the replicas stored at malicious nodes are always under attack (recall that up to  $cr - 1$  out of  $R$  file replicas could be unavailable or corrupted).

Hence, the fact that a malicious replica holder incorrectly implements file read/write operation or that the adversary is aware of the tokens of those file replicas stored at malicious nodes does not harm the system. Also, by design, an adversary who knows one token  $rlt^i$  for replica  $f^i$  would not be able to guess the file name  $f$  or its location key  $lk$  or the tokens for others replicas of file  $f$  (see Section 2.7).

**Adversary.** An adversary cannot access any replica of file  $f$  stored at a good node simply because it cannot guess the token  $rlt^i$  without knowing its location key. However, when a good node is compromised an adversary would be able to directly obtain the tokens for all files stored at that node. In general, an adversary could compile a list of tokens as it compromises good nodes, and corrupt the file replicas corresponding to these tokens at any later point in time. Eventually, the adversary would succeed in corrupting  $cr$  or more replicas of a file  $f$  without knowing its location key. LocationGuard addresses such attacks using a location rekeying technique discussed in Section 2.9.3.

In the subsequent sections, we show how to generate a replica location token  $rlt^i$  ( $1 \leq i \leq R$ ) from a file  $f$  and its location key (Section 2.7), and how the lookup algorithm  $\Psi$  performs a lookup on a pseudo-filename  $rlt^i$  without revealing the capability  $rlt^i$  to malicious nodes in the overlay network (Section 2.8). It is important to note that the ability to guard the lookup from attacks like eavesdropping is critical to the file location hiding scheme, since a lookup operation (using a lookup protocol such as Chord) on identifier  $rlt^i$  typically proceeds in plain-text through a sequence of nodes on the overlay network. Hence, an adversary may collect file tokens by simply sniffing lookup queries over the overlay network. The adversary could use these stolen file tokens to perform write operations on the corresponding file replicas, and thus corrupt them, without the knowledge of their location keys.

## 2.7 Location Keys

The first and most simplistic component of LocationGuard is the concept of location keys. The design of location key needs to address the following two questions: (1) How to choose a location key? (2) How to use a location key to generate a replica location token – the capability to access a file replica?

The first step in designing location keys is *to determining the type of string used as the identifier of a location key*. Let user  $u$  be the owner of a file  $f$ . User  $u$  should choose a long random bit string (128-bits)  $lk$  as the location key for file  $f$ . The location key  $lk$  should be hard to guess. For example, the key  $lk$  should not be semantically attached to or derived from the file name ( $f$ ) or the owner name ( $u$ ).

The second step is *to find a pseudo-random function* to derive the replica location tokens  $rlt^i$  ( $1 \leq i \leq R$ ) from the filename  $f$  and its location key  $lk$ . The pseudo-filename  $rlt^i$  is used as a file replica identifier to locate the  $i^{th}$  replica of file  $f$  on the overlay network. Let  $E_{lk}(x)$  denote a keyed pseudo-random function with input  $x$  and a secret key  $lk$  and  $\parallel$  denotes string concatenation. We derive the location token  $rlt^i = E_{lk}(f \parallel i)$ . Given a replica's identifier  $rlt^i$ , one can use the lookup protocol  $\Psi$  to locate it on the overlay network. The function  $E$  should satisfy the following conditions:

- 1a) Given  $(f \parallel i)$  and  $lk$  it is easy to compute  $E_{lk}(f \parallel i)$ .
- 2a) Given  $(f \parallel i)$  it is hard to guess  $E_{lk}(f \parallel i)$  without knowing  $lk$ .
- 2b) Given  $E_{lk}(f \parallel i)$  it is hard to guess the file name  $f$ .
- 2c) Given  $E_{lk}(f \parallel i)$  and  $f$  it is hard to guess  $lk$ .

Condition 1a ensures that it is very easy for a valid user to locate a file  $f$  as long as it is aware of the file's location key  $lk$ . Condition 2a states that it should be very hard for an adversary to guess the location of a target file  $f$  without knowing its location key. Condition 2b ensures that even if an adversary obtains the identifier  $rlt^i$  of replica  $f^i$ , he/she cannot deduce the file name  $f$ . Finally, Condition 2c requires that even if an adversary obtains the identifiers of one or more replicas of file  $f$ , he/she would not be able to derive the location key  $lk$  from them. Hence, the adversary still has no clue about the remaining replicas of the file  $f$  (by Condition 2a). Conditions 2b and 2c play an important role in ensuring good location hiding property. This is because for any given file  $f$ , some of the replicas of file  $f$  could be stored at malicious nodes. Thus an adversary could be aware of some of the replica identifiers. Finally, observe that Condition 1a and Conditions {2a, 2b, 2c} map to

Property 1 and Property 2 in Definition 1 (in Section 2.6.3) respectively.

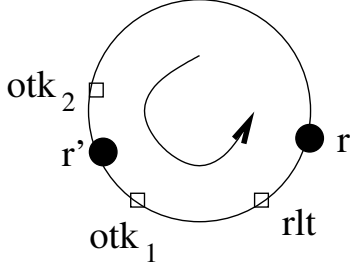
There are a number of cryptographic tools that satisfies our requirements specified in Conditions 1a, 2a, 2b and 2c. Some possible candidates for the function  $E$  are (i) a keyed-hash function like HMAC-MD5 [52], (ii) a symmetric key encryption algorithm like DES [36] or AES [63], and (iii) a PKI based encryption algorithm like RSA [85]. We chose to use a keyed-hash function like HMAC because it can be computed very efficiently. HMAC-MD5 computation is about 40 times faster than AES encryption and about 1000 times faster than RSA encryption using the standard OpenSSL library [64]. In the remaining part of this chapter, we use *khash* to denote a keyed pseudo-random function that is used to derive a file’s replica location tokens from its name and its secret location key.

## 2.8 Routing guard

The second and fundamental component of LocationGuard is the routing guard. The design of routing guard aims at securing the lookup of file  $f$  such that it will be very hard for an adversary to obtain the replica location tokens by eavesdropping on the overlay network. Concretely, let  $rlt^i$  ( $1 \leq i \leq R$ ) denote a replica location token derived from the file name  $f$ , the replica number  $i$ , and  $f$ ’s location key  $lk$ . We need to secure the lookup algorithm  $\Psi_{lk}(rlt^i)$  such that the lookup on pseudo-filename  $rlt^i$  does not reveal the capability  $rlt^i$  to other nodes on the overlay network. Note that a file’s capability  $rlt^i$  does not reveal the file’s name; but it allows an adversary to write on the file and thus corrupt it (see reference file system in Section 2.6.4).

There are two possible approaches to implement a secure lookup algorithm: (1) centralized approach and (2) decentralized approach. In the centralized approach, one could use a trusted location server [47] to return the location of any file on the overlay network. However, such a location server would become a viable target for DoS and host compromise attacks.

In this section, we present a decentralized secure lookup protocol that is built on top of the Chord protocol. Note that a naive Chord-like lookup protocol  $\Gamma(rlt^i)$  cannot be directly used because it reveals the token  $rlt^i$  to other nodes on the overlay network.



**Figure 8:** Lookup Using File Identifier Obfuscation: Illustration

$1 - pr_{sq}$	$2^{-10}$	$2^{-15}$	$2^{-20}$	$2^{-25}$	$2^{-30}$
$srg$	$2^{98}$	$2^{93}$	$2^{88}$	$2^{83}$	$2^{78}$
$E[retries]$	$2^{-10}$	$2^{-15}$	$2^{-20}$	$2^{-25}$	$2^{-30}$
hardness (years)	$2^{38}$	$2^{33}$	$2^{28}$	$2^{23}$	$2^{18}$

**Table 3:** Lookup Identifier obfuscation

### 2.8.1 Overview

The fundamental idea behind the routing guard is as follows. Given a file  $f$ 's location key  $lk$  and replica number  $i$ , we want to find a safe region in the identifier space where we can obtain a huge collection of *obfuscated tokens*, denoted by  $\{OTK^i\}$ , such that, with high probability,  $\Gamma(otk^i) = \Gamma(rlt^i)$ ,  $\forall otk^i \in OTK^i$ . We call  $otk^i \in OTK^i$  an obfuscated identifier of the token  $rlt^i$ . Each time a user  $u$  wishes to lookup a token  $rlt^i$ , it performs a lookup on some randomly chosen token  $otk^i$  from the obfuscated identifier set  $OTK^i$ . Routing guard ensures that even if an adversary were to observe obfuscated identifiers from the set  $OTK^i$  for one full year, it would be highly infeasible for the adversary to guess the token  $rlt^i$ .

We now describe the concrete implementation of the routing guard. For the sake of simplicity, we assume a unit circle for the Chord's identifier space; that is, node identifiers and file identifiers are real values from 0 to 1 that are arranged on the Chord ring in the anti-clockwise direction. Let  $ID(r)$  denote the identifier of node  $r$ . If  $r$  is the destination node of a lookup on file identifier  $rlt^i$ , i.e.,  $r = \Gamma(rlt^i)$ , then  $r$  is the node that immediately succeeds  $rlt^i$  in the anti-clockwise direction on the Chord ring. Formally,  $r = \Gamma(rlt^i)$  if  $ID(r) \geq rlt^i$  and there exists no other nodes, say  $v$ , on the Chord ring such that  $ID(r) > ID(v) \geq rlt^i$ .

We first introduce the concept of *safe obfuscation* to guide us in finding an obfuscated identifier set  $OTK^i$  for a given replica location token  $rlt^i$ . We say that an obfuscated identifier  $otk^i$  is a safe obfuscation of identifier  $rlt^i$  if and only if a lookup on both  $rlt^i$  and  $otk^i$  result in the same physical node  $r$ . For example, in Figure 8, identifier  $otk_1^i$  is a safe

obfuscation of identifier  $rlt^i$  ( $\Gamma(rlt^i) = \Gamma(otk_1^i) = r$ ), while identifier  $otk_2^i$  is unsafe ( $\Gamma(otk_2^i) = r' \neq r$ ).

We define the set  $OTK^i$  as a set of all identifiers in the range  $(rlt^i - srg, rlt^i)$ , where  $srg$  denotes a safe obfuscation range ( $0 \leq srg < 1$ ). When a user intends to query for a replica location token  $rlt^i$ , the user actually performs a lookup on an obfuscated identifier  $otk^i = obfuscate(rlt^i) = rlt^i - random(0, srg)$ . The function  $random(0, srg)$  returns a number chosen uniformly and randomly in the range  $(0, srg)$ .

We choose a safe value  $srg$  such that:

- (C1) With high probability, any obfuscated identifier  $otk^i$  is a safe obfuscation of the token  $rlt^i$ .
- (C2) Given a large collection of obfuscated identifiers  $\{otk^i\}$  it is very hard for an adversary to guess the actual identifier  $rlt^i$ .

Note that if  $srg$  is too small condition C1 is more likely to hold, while condition C2 is more likely to fail. In contrast, if  $srg$  is too big, condition C2 is more likely to hold but condition C1 is more likely to fail. In our first prototype development of LocationGuard, we introduce a system defined parameter  $pr_{sq}$  to denote the minimum probability that any obfuscation is required to be safe. In the subsequent sections, we present a technique to derive  $srg$  as a function of  $pr_{sq}$ . This permits us to quantify the tradeoff between condition C1 and condition C2.

## 2.8.2 Determining the Safe Obfuscation Range

Observe from Figure 8 that a obfuscation  $rand$  on identifier  $rlt^i$  is safe if  $rlt^i - rand > ID(r')$ , where  $r'$  is the immediate predecessor of node  $r$  on the Chord ring. Thus, we have  $rand < rlt^i - ID(r')$ . The expression  $rlt^i - ID(r')$  denotes the distance between identifiers  $rlt^i$  and  $ID(r')$  on the Chord identifier ring, denoted by  $dist(rlt^i, ID(r'))$ . Hence, we say that a obfuscation  $rand$  is safe with respect to identifier  $rlt^i$  if and only if  $rand < dist(rlt^i, ID(r'))$ , or equivalently,  $rand$  is chosen from the range  $(0, dist(rlt^i, ID(r')))$ .

We use Theorem 2.8.1 to show that  $\Pr(dist(rlt^i, ID(r')) > x) = e^{-x*N}$ , where  $N$  denotes the number of nodes on the overlay network and  $x$  denotes any value satisfying

$0 \leq x < 1$ . Informally, the theorem states that the probability that the predecessor node  $r'$  is further away from the identifier  $rlt^i$  decreases exponentially with the distance. Since an obfuscation  $rand$  is safe with respect to  $rlt^i$  if  $dist(rlt^i, ID(r')) > rand$ , the probability that a obfuscation  $rand$  is safe can be calculated using  $e^{-rand*N}$ .

Now, one can ensure that the minimum probability of any obfuscation being safe is  $pr_{sq}$  as follows. We first use  $pr_{sq}$  to obtain an upper bound on  $rand$ : By  $e^{-rand*N} \geq pr_{sq}$ , we have,  $rand \leq \frac{-\log_e(pr_{sq})}{N}$ . Hence, if  $rand$  is chosen from a safe range  $(0, srg)$ , where  $srg = \frac{-\log_e(pr_{sq})}{N}$ , then all obfuscations are guaranteed to be safe with a probability greater than or equal to  $pr_{sq}$ .

For instance, when we set  $pr_{sq} = 1 - 2^{-20}$  and  $N = 1$  million nodes,  $srg = -\frac{\log_e(pr_{sq})}{N} = 2^{-40}$ . Hence, on a 128-bit Chord ring  $rand$  could be chosen from a range of size  $srg = 2^{128} * 2^{-40} = 2^{88}$ . Table 3 shows the size of a  $pr_{sq}$ -safe obfuscation range  $srg$  for different values of  $pr_{sq}$ . Observe that if we set  $pr_{sq} = 1$ , then  $srg = -\frac{\log_e(pr_{sq})}{N} = 0$ . Hence, if we want 100% safety, the obfuscation range  $srg$  must be zero, i.e., the token  $rlt^i$  cannot be obfuscated.

**Theorem 2.8.1** *Let  $N$  denote the total number of nodes in the system. Let  $dist(x, y)$  denote the distance between two identifiers  $x$  and  $y$  on a Chord's unit circle. Let node  $r'$  be the node that is the immediate predecessor for an identifier  $rlt^i$  on the anti-clockwise unit circle Chord ring. Let  $ID(r')$  denote the identifier of the node  $r'$ . Then, the probability that the distance between identifiers  $rlt^i$  and  $ID(r')$  exceeds  $rg$  is given by  $Pr(dist(rlt^i, ID(r')) > x) = e^{-x*N}$  for some  $0 \leq x < 1$ .*

**Proof** Let  $Z$  be a random variable that denotes the distance between an identifier  $rlt^i$  and node  $r'$ . Let  $f_Z(x)$  denote the probability distribution function (pdf) that the node  $r'$  is at a distance  $x$  from the identifier  $rlt^i$ , i.e.,  $dist(ID(r'), rlt^i) = x$ . We first derive the probability distribution  $f_Z(x)$  and use it to compute  $Pr(Z > x) = Pr(dist(rlt^i, ID(r')) > x)$ .

By the uniform and random distribution properties of the hash function the identifier of a node will be uniformly and randomly distributed between  $(0, 1)$ . Hence, the probability that the identifier of any node falls in a segment of length  $x$  is equal to  $x$ . Hence, with

probability  $\Delta x$ , a given node exists between a distance of  $(x, x + \Delta x)$  from the identifier  $rlt^i$  (for any arbitrarily small region  $\Delta x$ ). When there are  $N$  nodes in the system, the probability that one of them exists between a distance  $(x, x + \Delta x)$  is  $N * \Delta x$ . Similarly, the probability that none of other node  $N - 1$  nodes lie within a distance  $rg$  from identifier  $rlt^i$  is  $(1 - x)^{N-1}$ . Therefore,  $f_Z(x)$  is given by Equation 8.

$$f_Z(x) = N * (1 - x)^{N-1} \quad (8)$$

Now, using the probability density function in Equation 8 one can derive the cumulative distribution function (cdf),  $\Pr(Z > x) = (1 - x)^N \approx e^{-x*N}$  (for small values of  $x$ ) using standard techniques in probability theory. ■

### 2.8.3 Ensuring Safe Obfuscation

Given that when  $pr_{sq} < 1$ , there is small probability that an obfuscated identifier is not safe, i.e.,  $1 - pr_{sq} > 0$ . We first discuss the motivation for detecting and repairing unsafe obfuscations and then describe how to guarantee good safety by our routing guard through a self-detection and self-healing process.

Let node  $r$  be the result of a lookup on identifier  $rlt^i$  and node  $v$  ( $v \neq r$ ) be the result of a lookup on an unsafe obfuscated identifier  $otk^i$ . To perform a file read/write operation after locating the node that stores the file  $f$ , the user has to present the location token  $rlt^i$  to node  $v$ . If a user does not check for unsafe obfuscation, then the file token  $rlt^i$  would be exposed to some other node  $v \neq r$ . If node  $v$  were malicious, then it could misuse this information to corrupt the file replica actually stored at node  $r$  (using the capability  $rlt^i$ ).

We require a user to verify whether an obfuscated identifier is safe or not using the following check: An obfuscated identifier  $otk^i$  is considered *safe* if and only if  $rlt^i \in (otk^i, ID(v))$ , where  $v = \Gamma(otk^i)$ . By the definition of  $v$  and  $otk^i$ , we have  $otk^i \leq ID(v)$  and  $otk^i \leq rlt^i$  ( $rand \geq 0$ ). By  $otk^i \leq rlt^i \leq ID(v)$ , node  $v$  should be the immediate successor of the identifier  $rlt^i$  and thus be responsible for it. If the check failed, i.e.,  $rlt^i > ID(v)$ , then node  $v$  is definitely not a successor of the identifier  $rlt^i$ . Hence, the user can flag  $otk^i$  as an unsafe obfuscation of  $rlt^i$ . For example, referring Figure 8,  $otk_1^i$  is safe because,  $rlt^i \in$

$(otk_1^i, ID(r))$  and  $r = \Gamma(otk_1^i)$ , and  $otk_2^i$  is unsafe because,  $rlt^i \notin (otk_2^i, ID(r'))$  and  $r' = \Gamma(otk_2^i)$ .

When an obfuscated identifier is flagged as unsafe, the user needs to retry the lookup operation with a new obfuscated identifier. This retry process continues until *max\_retries* rounds or until a safe obfuscation is found. Thanks to the fact that the probability of an unsafe obfuscation can be extremely small, the call for retry rarely happens. We also found from our experiments that the number of retries required is almost always zero and seldom exceeds one. We believe that using *max\_retries* equal to two would suffice even in a highly conservative setting. Table 3 shows the expected number of retries required for a lookup operation for different values of  $pr_{sq}$ .

#### 2.8.4 Strength of Routing guard

The strength of a routing guard refers to its ability to counter lookup sniffing based attacks. A typical lookup sniffing attack is called the *range sieving attack*. Informally, in a range sieving attack, an adversary sniffs lookup queries on the overlay network, and attempts to deduce the actual identifier  $rlt^i$  from its multiple obfuscated identifiers. We show that an adversary would have to expend  $2^{28}$  years to discover a replica location token  $rlt^i$  even if it has observed  $2^{25}$  obfuscated identifiers of  $rlt^i$ . Note that  $2^{25}$  obfuscated identifiers would be available to an adversary if the file replica  $f^i$  was accessed once a second for one full year by some legal user of the file  $f$ .

One can show that given multiple obfuscated identifiers it is non-trivial for an adversary to categorize them into groups such that all obfuscated identifiers in a group are actually obfuscations of one identifier. To simplify the description of a range sieving attack, we consider the worst case scenario where an adversary is capable of categorizing obfuscated identifiers (say, based on their numerical proximity).

We first concretely describe the range sieving attack assuming that  $pr_{sq}$  and  $srg$  (from Theorem 2.8.1) are publicly known. When an adversary obtains an obfuscated identifier  $otk^i$ , the adversary knows that the actual capability  $rlt^i$  is definitely within the range  $RG = (otk^i, otk^i + srg)$ , where  $(0, srg)$  denotes a  $pr_{sq}$ -safe range. In fact, if obfuscations

are uniformly and randomly chosen from  $(0, srg)$ , then given an obfuscated identifier  $otk^i$ , the adversary knows *nothing more* than the fact that the actual identifier  $rlt^i$  could be uniformly and randomly distributed over the range  $RG = (otk^i, otk^i + srg)$ . However, if a persistent adversary obtains multiple obfuscated identifiers  $\{otk_1^i, otk_2^i, \dots, otk_{nid}^i\}$  that belong to the same target file, the adversary can *sieve* the identifier space as follows. Let  $RG_1, RG_2, \dots, RG_{nid}$  denote the ranges corresponding to  $nid$  random obfuscations on the identifier  $rlt^i$ . Then the capability of the target file is guaranteed to lie in the sieved range  $RG_s = \cap_{j=1}^{nid} RG_j$ . Intuitively, if the number of obfuscated identifiers ( $nid$ ) increases, the size of the sieved range  $RG_s$  decreases. For all tokens  $tk \in RG_s$ , the likelihood that the obfuscated identifiers  $\{otk_1^i, otk_2^i, \dots, otk_{nid}^i\}$  are obfuscations of the identifier  $tk$  is equal. In fact, the probability of observing  $otk_j^i$  for some  $1 \leq j \leq nid$  given that the actual token is  $tk$  is  $Pr(otk_j^i | tk) = \frac{1}{srg}, \forall tk \in RG_s$ . Also, the probability of observing the obfuscated identifiers  $\{otk_1^i, otk_2^i, \dots, otk_{nid}^i\}$  given that the actual token is  $tk$  is  $Pr(\{otk_1^i, otk_2^i, \dots, otk_{nid}^i\} | tk) = \frac{1}{srg C_{nid}^{srg}}, \forall tk \in RG_s$ . Note that  ${}^{srg}C_{nid}$  denotes the number of ways of choosing  $nid$  balls from a pool of  $srg$  non-identical balls. Hence, the adversary is left with no smart strategy for searching the sieved range  $RG_s$  other than performing a brute force attack on some random enumeration of identifiers  $tk \in RG_s$ .

Let  $E[RG_s]$  denote the expected size of the sieved range. Theorem 2.8.2 shows that  $E[RG_s] = \frac{srg}{nid}$ . Hence, if the safe range  $srg$  is significantly larger than  $nid$  then the routing guard can tolerate the range sieving attack. Recall the example in Section 2.8 where  $pr_{sq} = 1 - 2^{-20}$ ,  $N = 10^6$ , the safe range  $srg = 2^{88}$ . Suppose that a target file is accessed once per second for one year; this results in  $2^{25}$  file accesses. An adversary who logs all obfuscated identifiers over a year could sieve the range to about  $E[|RG_s|] = 2^{63}$ . Assuming that the adversary performs a brute force attack on the sieved range, by attempting a file read operation at the rate of one read per millisecond, the adversary would have tried  $2^{35}$  read operations per year. Thus, it would take the adversary about  $2^{63}/2^{35} = 2^{28}$  years to discover the actual file identifier. Table 3 summarizes the hardness of breaking the obfuscation scheme for different values of  $pr_{sq}$  (minimum probability of safe obfuscation), assuming that the adversary has logged  $2^{25}$  file accesses (one access per second for one year)

and that the nodes permit at most one file access per millisecond.

**Discussion.** An interesting observation follows from the above discussion: the amount of time taken to break the file identifier obfuscation technique is almost independent of the number of attackers. This is a desirable property. It implies that as the number of attackers increases in the system, the hardness of breaking the file capabilities will not decrease. The reason for location key based systems to have this property is because the time taken for a brute force attack on a file identifier is fundamentally limited by the rate at which a hosting node permits accesses on files stored locally. On the contrary, a brute force attack on a cryptographic key is inherently parallelizable and thus becomes more powerful as the number of attackers increases.

**Theorem 2.8.2** *Let  $nid$  denote the number of obfuscated identifiers that correspond to a target file. Let  $RG_s$  denote the sieved range using the range sieving attack. Let  $srg$  denote the maximum amount of obfuscation that could be  $pr_{sq}$ -safely added to a file identifier. Then, the expected size of range  $RG_s$  can be calculated by  $E[|RG_s|] = \frac{srg}{nid}$ .*

**Proof** Let  $otk_{min}^i = rlt^i - rand_{max}$  and  $otk_{max}^i = rlt^i - rand_{min}$  denote the minimum and the maximum value of an obfuscated identifier that has been obtained by an adversary, where  $rand_{max}$  and  $rand_{min}$  are chosen from the safe range  $(0, srg)$ . Then, we have the sieved range  $RG_s = (otk_{max}^i, otk_{min}^i + srg)$ , namely, from the highest lower bound to the lowest upper bound. The sieved range  $RG_s$  can be partitioned into two ranges  $RG_{min}$  and  $RG_{max}$ , where  $RG_{min} = (otk_{max}^i, rlt^i)$  and  $RG_{max} = (rlt^i, otk_{min}^i + srg)$ . Thus we have  $E[|RG_s|] = E[|RG_{min}|] + E[|RG_{max}|]$ .

The size of the range  $RG_{min}$ , denoted as  $|RG_{min}|$ , equals to  $rand_{min}$  since is  $rlt^i - otk_{max}^i = rand_{min}$ . We show that the cumulative distribution function of  $rand_{min}$  is given by Equation 9.

$$Pr(rand_{min} > rg) = \left(1 - \frac{rg}{srg}\right)^{nid} \quad (9)$$

Since an obfuscation  $rand$  is chosen uniformly and randomly over a range  $(0, srg)$ , for  $0 \leq rg \leq srg$ , the probability that any obfuscation  $rand$  is smaller than  $rg$ , denoted by  $Pr(rand \leq rg)$ , is  $\frac{rg}{srg}$ . Hence, the probability that any obfuscation  $rand$  is greater than  $rg$

is  $Pr(rand > rg) = 1 - Pr(rand \leq rg) = 1 - \frac{rg}{srg}$ . Now we compute the probability that  $rand_{min} = \min\{rand_1, rand_2, \dots, rand_{nid}\}$  is greater than  $rg$ . We have  $Pr(rand_{min} > rg) = Pr((rand_1 > rg) \wedge (rand_2 > rg) \wedge \dots \wedge (rand_{nid} > rg)) = \prod_{j=1}^{nid} Pr(rand_j > rg) = \left(1 - \frac{rg}{srg}\right)^{nid}$ .

Now, using standard techniques from probability theory and Equation 9, one can derive the expected value of  $rand_{min}$ :  $E[|RG_{min}|] = E[rand_{min}] \approx \frac{srg}{nid}$ . Symmetrically, one can show that the expected size of range  $RG_{max}$  is  $E[|RG_{max}|] \approx \frac{srg}{nid}$ . Hence the expected size of sieved range is  $E[|RG_s|] = E[|RG_{min}|] + E[|RG_{max}|] \geq \frac{srg}{nid}$ . ■

## 2.9 Location Inference Guards

Location inference attacks refer to those attacks wherein an adversary attempts to infer the location of a file using *indirect* techniques that exploit file metadata information such as file access frequency, file size, and so forth. LocationGuard includes a suite of four fundamental and inexpensive inference guards: lookup frequency inference guard, end-user IP-address inference guard, file replica inference guard and file size inference guard. LocationGuard also includes a capability revocation based location rekeying mechanism as a general guard against any inference attack. In this section, we present the four fundamental inference guards and the location rekeying technique in detail.

### 2.9.1 Passive Inference Guards

Passive inference attacks refer to those attacks wherein an adversary attempts to infer the location of a target file by passively observing the overlay network. We present two inference guards: lookup frequency inference guard and end-user IP-address inference guard to guard the file system against two common passive inference attacks. The lookup frequency inference attack is based on the ability of malicious nodes to observe the frequency of lookup queries on the overlay network. Assuming that the adversary knows the relative file popularity, it can use the target file's lookup frequency to infer its location. The end-user IP-address inference attack is based on assumption that the identity of the end-user can be inferred from its IP-address by an overlay network node  $r$ , when the user requests

node  $r$  to perform a lookup on its behalf. The malicious node  $r$  could log and report this information to the adversary.

### 2.9.1.1 Lookup Frequency Inference Guard

In this section we present lookup frequency inference attack that would help a strategic adversary to infer the location of a target file on the overlay network. It has been observed that the general popularity of the web pages accessed over the Internet follows a Zipf-like distribution [119]. An adversary may study the frequency of file accesses by sniffing lookup queries and match the observed file access frequency profile with a actual (pre-determined) frequency profile to infer the location of a target file <sup>4</sup>. Note that if the frequency profile of the files stored in the file system is flat (all files are accessed with the same frequency) then an adversary will not be able to infer any information. Lemma 2.9.1 formalizes the notion of perfectly hiding a file from a frequency inference attack.

**Lemma 2.9.1** *Let  $F$  denote the collection of files in the file system. Let  $\lambda'_f$  denote the apparent frequency of accesses to file  $f$  as perceived by an adversary. Then, the collection of files is perfectly hidden from frequency inference attack if  $\lambda'_f = c: \forall f \in F$  and some constant  $c$ .*

**Corollary 2.9.2** *A collection of read-only files can be perfectly hidden from frequency inference attack.*

**Proof** Let  $\lambda_f$  denote the actual frequency of accesses on a file  $f$ . Set the number replicas for file  $f$  to be proportional to its access frequency, namely  $R_f = \frac{1}{c} * \lambda_f$  (for some constant  $c > 0$ ). When a user wishes to read the file  $f$ , the user randomly chooses one replica of file  $f$  and issues a lookup query on it. From an adversary's point of view it would seem that the access frequency to all the file replicas in the system is identical, namely,  $\forall f \lambda'_{fi} = \frac{\lambda_f}{R_f} = c$  ( $1 \leq i \leq R_f$  for file  $f$ ). By Lemma 2.9.1, an adversary would not be able to derive any useful information from a frequency inference attack. ■

---

<sup>4</sup>This is analogous to performing a frequency analysis attack on old symmetric key ciphers like the Caesar's cipher [58]

Interestingly, the replication strategy used in Corollary 2.9.2 improves the performance and load balancing aspect of the file system as well. However, it is not applicable to read-write files since an update operation on a file may need to update all the replicas of a file. In the following portions of this section, we propose two techniques to flatten the *apparent* frequency profile of read/write files.

**Guard by Result Caching.** The first technique to mitigate the frequency inference attack is to obfuscate the apparent file access frequency with lookup *result caching*. Lookup result caching, as the name indicates, refers to caching the results of a lookup query. Recall that wide-area network file systems like CFS, Farsite and OceanStore permit nodes to join and leave the overlay network. Let us for now consider only node departures. Consider a file  $f$  stored at node  $n$ . Let  $\lambda_f$  denote the rate at which users accesses the file  $f$ . Let  $\mu_{dep}$  denote the rate at which a node leaves the overlay network (rates are assumed to be exponentially distributed). The first time the user accesses the file  $f$ , the lookup result (namely, node  $n$ ) is cached. The lookup result is implicitly invalidated when the user attempts to access file  $f$  the first time after node  $n$  leaves the overlay network. When the lookup result is invalidated, the user issues a fresh lookup query for file  $f$ . One can show that the apparent frequency of file access as observed by an adversary is  $\lambda'_f = \frac{\lambda_f \mu_{dep}}{\lambda_f + \mu_{dep}}$  (assuming exponential distribution for  $\lambda_f$  and  $\mu_{dep}$ ). The probability that any given file access results is a lookup is equal to the probability that the node responsible for the file leaves before the next access and is given by  $Pr_{lookup} = \frac{\mu_{dep}}{\lambda_f + \mu_{dep}}$ . Hence, the apparent file access frequency is equal to the product of the actual file access frequency ( $\lambda_f$ ) and the probability that a file access results in a lookup operation ( $Pr_{lookup}$ ). Intuitively, in a static scenario where nodes never leave the network ( $\mu_{dep} \ll \lambda_f$ ),  $\lambda'_f \approx \mu_{dep}$ ; and when nodes leave the network very frequently ( $\mu_{dep} \gg \lambda_f$ ),  $\lambda'_f \approx \lambda_f$ . Hence, more static the overlay network is, harder it is for an adversary to perform a frequency inference attack since it would appear as if all files in the system are accessed at an uniform frequency  $\mu_{dep}$ .

It is very important to note that a node  $m$  storing a file  $f$  may infer  $f$ 's name since the user has to ultimately access node  $m$  to operate on file  $f$ . Hence, an adversary may infer the identities of files stored at malicious nodes. However, it would be very hard for an

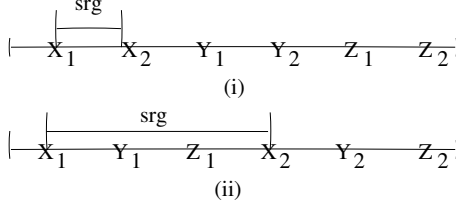
adversary to infer the identities of files stored at good nodes.

**Guard by File Identifier Obfuscation.** The second technique that makes the frequency inference attack harder is based on the file identifier obfuscation technique described in Section 2.8. Let  $f_1, f_2, \dots, f_{nf}$  denote the files stored at some node  $n$ . Let the identifiers of these replicas be  $rlt_1, rlt_2, \dots, rlt_{nf}$ . Let the target file be  $f_1$ . The key idea is to obfuscate the identifiers such that an adversary would not be able to distinguish between an obfuscated identifier intended for locating file  $f_1$  and that for some other file  $f_j$  ( $2 \leq j \leq nf$ ) stored at node  $n$ .

More concretely, when a user performs a lookup for  $f_1$ , the user would choose some random identifier in the range  $R_1 = (rlt_1 - srg, rlt_1)$ . A clever adversary may *cluster* identifiers based on their numerical closeness and perform a frequency inference attack on these clusters. However, one could defend the system against such a clustering technique by appropriately choosing a safe obfuscation range. Figure 9 presents the key intuition behind this idea diagrammatically. As the range  $R_1$  overlaps with the ranges of more and more files stored at node  $n$ , the clustering technique and consequently the frequency inference attack would perform poorly. Let  $R_1 \cap R_2$  denote the set of identifiers that belongs the intersection of ranges  $R_1$  and  $R_2$ . Then, given an identifier  $otk \in R_1 \cap R_2$ , an adversary would not be able to distinguish whether the lookup was intended for file  $f_1$  or  $f_2$ ; but the adversary would definitely know that the lookup was intended either for file  $f_1$  or  $f_2$ . Observe that amount of information inferred by an adversary becomes poorer and poorer as more and more ranges overlap. Also, as the number of files ( $nf$ ) stored at node  $n$  increases, even a small obfuscation might introduce significant overlap between the ranges of different files stored at node  $n$ .

The apparent access frequency of a file  $f$  is computed as a weighted sum of the actual access frequencies of all files that share their range with file  $f$ . For instance, the apparent access frequency of file  $f_1$  (see Figure 9) is given by Equation 10.

$$\lambda'_{f_1} = \frac{X_1 Y_1 * \lambda_{f_1} + Y_1 Z_1 * \left(\frac{\lambda_{f_1} + \lambda_{f_2}}{2}\right) + Z_1 X_2 * \left(\frac{\lambda_{f_1} + \lambda_{f_2} + \lambda_{f_3}}{3}\right)}{srg} \quad (10)$$



**Figure 9:** Countering Frequency Analysis Attack by file identifier obfuscation.  $X_1X_2$ ,  $Y_1Y_2$  and  $Z_1Z_2$  denote the ranges of the obfuscated identifiers of files  $f_1$ ,  $f_2$ ,  $f_3$  stored at node  $n$ . Frequency inference attacks works in scenario (i), but not in scenario (ii). Given an identifier  $otk \in Y_1Z_1$ , it is hard for an adversary to guess whether the lookup was for file  $f_1$  or  $f_2$ .

The apparent access frequency of a file evens out the sharp variations between the frequencies of different files stored at a node, thereby making frequency inference attack significantly harder. We discuss more on how to quantify the effect of file identifier obfuscation on frequency inference attack in our experimental section 2.10.

#### 2.9.1.2 End-User IP-Address Inference Guard

In this section, we describe an end-user IP-address inference attack that assumes that the identity of an end-user can be inferred from his/her IP-address. Note that this is a worst-case-assumption; in most cases it may not possible to associate a user with one or a small number IP-addresses. This is particularly true if the user obtains IP-address dynamically (DHCP [31]) from a large ISP (Internet Service Provider).

A user typically locate their files on the overlay network by issuing a lookup query to some node  $r$  on the overlay network. If node  $r$  were malicious then it may log the file identifiers looked up by a user. Assuming that a user accesses only a small subset of the total number of files on the overlay network (including the target file) the adversary can narrow down the set of nodes on the overlay network that may potentially hold the target file. One possible solution is for users to issue lookup queries through a trusted *anonymizer*. The anonymizer accepts lookup queries from users and dispatches it to the overlay network without revealing the user’s IP-address. However, the anonymizer could itself become a viable target for the adversary.

A more promising solution is for the user to join the overlay network (just like other

nodes hosting files on the overlay network). When the user issues lookup queries, it is routed through some of its neighbors; if some of its neighbors are malicious, then they may log these lookup queries. However, it is non-trivial for an adversary to distinguish between the queries that *originated* at the user and those that were simply *routed* through it.

For the sake of simplicity, let us assume that  $q$  denotes the number of lookups issued per user per unit time. Assuming there are  $N$  users, the total lookup traffic is  $Nq$  lookups per unit time. Each lookup on an average requires  $\frac{1}{2} \log_2 N$  hops on Chord. Hence, the total lookup traffic is  $Nq * \frac{1}{2} \log_2 N$  hops per unit time. By the design of the overlay network, the lookup traffic is uniformly shared among all nodes in the system. Hence the number of lookup queries (per unit time) routed through any node  $u$  is  $\frac{1}{N} * \frac{1}{2} q N \log_2 N = q * \frac{1}{2} \log_2 N$ . Therefore, the ratio of lookup queries that originate at a node to that routed through it is  $\frac{q}{q * \frac{1}{2} \log_2 N} = \frac{2}{\log_2 N}$ . For  $N = 10^6$ , this ratio is about 0.1, thereby making it hard for an adversary to selectively pick only those queries that originated at a particular node. Further, not all neighbors of a node are likely to be bad; hence, it is rather infeasible for an adversary to collect all lookup traffic flowing through an overlay node.

## 2.9.2 Host Compromise based Inference Guards

Host compromise based inference attacks require the adversary to perform an active host compromise attack before it can infer the location of a target file. We present two inference guards: file replica inference guard and file size inference guard to guard the file system against two common host compromise based inference attacks. The file replica inference attack attempts to infer the identity of a file from its contents. Note that an adversary can reach the contents of a file only after it compromises the replica holder (unless the replica holder is malicious). The file size inference attack attempts to infer the identity of a file from its size. If the sizes of files stored on the overlay network are sufficiently skewed, the file size could by itself be sufficient to identify a target file.

### 2.9.2.1 File Replica Inference Guard

Despite making the file capabilities and file access frequencies appear random to an adversary, the contents of a file could by itself reveal the identity of the file  $f$ . The file  $f$  could be

encrypted to rule out the possibility of identifying a file from its contents. Even when the replicas are encrypted, an adversary can exploit the fact that all the replicas of file  $f$  are identical. When an adversary compromises a good node, it can extract a list of identifier and file content pairs (or a hash of the file contents) stored at that node. Note that an adversary could perform a frequency inference attack on the replicas stored at malicious nodes and infer their filenames. Hence, if an adversary were to obtain the encrypted contents of one of the replicas of a target file  $f$ , it could examine the extracted list of identifiers and file contents to obtain the identities of other replicas. Once, the adversary has the locations of  $cr$  copies of a file  $f$ , the  $f$  could be attacked easily. This attack is especially more plausible on read-only files since their contents do not change over a long period of time. On the other hand, the update frequency on read-write files might guard them from the file replica inference attack.

We guard read-only files (and files updated very infrequently) by making their replicas non-identical; this is achieved by encrypting each replica with a different cryptographic key. We derive the cryptographic key for the  $i^{th}$  replica of file  $f$  using its location key  $lk$  as  $k^i = khash_{lk}(f \parallel i \parallel \text{'cryptkey'})$ . Further, if one uses a symmetric key encryption algorithm in cipher-block-chaining mode (CBC mode [63] [36]), then we could reduce the encryption cost by using the same cryptographic key, but a different initialization vector ( $iv$ ) for encrypting different file replicas:  $k^i = khash_{lk}(f \parallel \text{'cryptkey'})$  and  $iv^i = khash_{lk}(f \parallel i \parallel \text{'ivec'})$ .

We show in our experimental section that even a small update frequency on read-write files is sufficient to guard them the file replica inference attack. Additionally, one could also choose to encrypt read-write file replicas with different cryptographic keys (to make the replicas non-identical) to improve their resilience to file replica inference attack.

### 2.9.2.2 File Size Inference Guard

File size inference attack is based on the assumption that an adversary might be aware of the target file's size. Malicious nodes (and compromised nodes) report the size of the files stored at them to an adversary. If the size of files stored on the overlay network follows a skewed distribution, the adversary would be able to identify the target file (much like the lookup

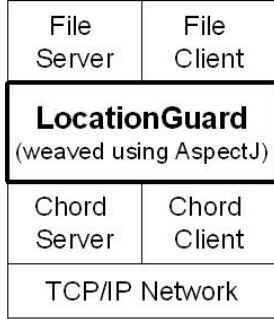
frequency inference attack). We guard the file system from this attack by fragmenting files into multiple file blocks of equal size. For instance, CFS divides files into blocks of 8 KBytes each and stores each file block separately. We hide the location of the  $j^{th}$  block in the  $i^{th}$  replica of file  $f$  using its location key  $lk$  and token  $rlt^{(i,j)} = khash_{lk}(f \parallel i \parallel j)$ . Note that the last file block may have to be padded to make its size 8 KBytes. Now, since all file blocks are of the same size, it would be vary hard for an adversary to perform file size inference attack. It is interesting to note that dividing files into blocks is useful in minimizing the communication overhead for small reads/writes on large files.

### 2.9.3 Location Rekeying

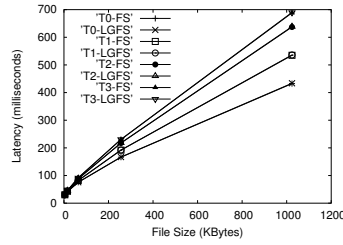
In addition to the inference attacks listed above, there could be other possible inference attacks on a LocationGuard based file system. In due course of time, the adversary might be able to gather enough information to infer the location of a target file. Location rekeying is a general defense against both *known and unknown* inference attacks. Users can periodically choose new location keys so as to render *all* past inferences made by an adversary *useless*. This is analogous to periodic rekeying of cryptographic keys. Unfortunately, rekeying is an expensive operation: rekeying cryptographic keys requires data to be re-encrypted; rekeying location keys requires files to be relocated on the overlay network. Hence, it is important to keep the rekeying frequency small enough to reduce performance overheads and large enough to secure files on the overlay network. In our experiments section, we estimate the periodicity with which location keys have to be changed in order to reduce the probability of an attack on a target file.

## 2.10 *Experimental Evaluation*

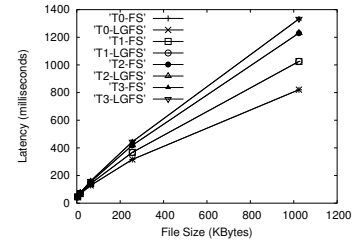
In this section, we report two sets of results. The first set of results is obtained from our prototype implementation of LocationGuard. The second of results is from simulation based experiments to evaluate the LocationGuard approach for building secure wide-area network file systems.



**Figure 10:** Implementation Architecture



**Figure 11:** File Read Overhead



**Figure 12:** File Write Overhead

File Type	T0	T1	T2	T3
Description	no cryptography	integrity only	confidentiality only	confidentiality and integrity

**Table 4:** LocationGuard File Types

### 2.10.1 Implementation-Based Experiments

In this section we briefly sketch our implementation of LocationGuard and quantify the overhead added by LocationGuard to the file system.

**Implementation.** We have implemented a prototype of LocationGuard on a publicly available Java code for the Chord lookup protocol [92]. We used AspectJ [33] to modify the Chord lookup protocol to include routing guard and lookup result caching. The method *obfuscate* implements lookup identifier obfuscation. The method *check\_safe\_obfuscation* implements our check for safe obfuscation; if the check fails then it calls *obfuscate* followed by the Chord lookup protocol. The AspectJ compiler statically weaves the *obfuscate* method and the *check\_safe\_obfuscation* method before and after all method calls to the Chord lookup protocol respectively.

The file system is implemented on top of the overlay network. We split files into blocks of 8KBytes and store each block at a location determined by the file’s location key. The file system is assumed to be flat (no directory hierarchy). The file names are simply the 32 Byte hexadecimal representation of the 128-bit file identifier. Access control in our system is implicit; if the file exists then the requested read/write operation is performed else an error is returned.

LocationGuard permits files to be any one of the four types: no cryptographic security (T0), integrity only (T1), confidentiality only (T2), and confidentiality and integrity (T3). To ensure file integrity, the file includes a keyed message authentication code using the HMAC-MD5 keyed hash function. To ensure file confidentiality, the file is encrypted using the AES-128 encryption algorithm. Finally, adding message authentication code (using MD5 [80] or SHA1 [32]) followed by encryption (using AES-128) guarantees both file confidentiality and integrity. We assume that the file owners distribute location keys and cryptographic keys through a secure out-of-band mechanism. Figure 10 shows our implementation architecture and Table 4 shows the four file types.

**Operational Overhead.** We ran our prototype implementation on eight machines each with 8-processors (550MHz Intel Pentium III Xeon processor running RedHat Linux 9.0) connected via a high speed LAN. In reality the nodes would be distributed on a wide-area network. However, we believe that this setup would be equally insightful in providing us the percentage overhead added by LocationGuard.

We first quantify the performance and storage overheads incurred by LocationGuard. Let us consider a typical file read/write operation. The operation consists of the following steps: (i) generate the replica location tokens, (ii) lookup the replica holders on the overlay network, and (iii) process the request at replica holders. Step (i) requires computations using the keyed-hash function with location keys, which otherwise would have required computations using a normal hash function. We found that the computation time difference between HMAC (a keyed pseudo-random function) and MD5 (a pseudo-random function) is negligibly small (order of a few microseconds) using the standard OpenSSL library [64]. Step (ii) involves a pseudo-random number generation (few microseconds using the OpenSSL library) and may require lookups to be retried in the event that the obfuscated identifier turns out to be unsafe. Given that unsafe obfuscations are extremely rare (see Table 3) retries are only required occasionally and thus this overhead is negligible. Step (iii) adds no overhead because our access check is almost free. As long as the user can present the correct pseudo-filename (token), the replica holder would honor a request on that file. Figures 11 and 12 shows the overhead of LocationGuard for file read and file write operations

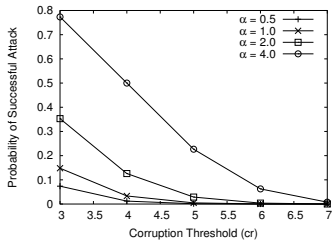
respectively. Each value reported in this experiment has been averaged over 64 runs. Note that file read/write operations of size greater than one block were parallelized, with each file block operation proceeding in parallel. Observe that the latency for file operations in a naive file system (FS) and LocationGuard (LGFS) is almost the same. For read operations maximum overhead due to LocationGuard was about 1.5ms (relative overhead of 0.4%) and that for write operation was 1.6ms (relative overhead of 0.3%).

Now, let us compare the storage overhead at the users and the nodes that are a part of the overlay network. Users need to store only an additional 128-bit location key (16 Bytes) along with other file meta-data for each file they want to access. Even a user who uses 1 million files on the overlay network needs to store only an additional 16MBytes of location keys. Further, there is no extra storage overhead on the rest of the nodes on the overlay network.

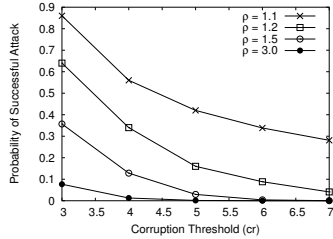
### 2.10.2 Simulation-Based Experiments

We implemented our simulator using a discrete event simulation [36] model. We simulate the Chord lookup protocol [106] on the overlay network comprising of  $N = 1024$  nodes. In all experiments reported in this chapter, a random  $p = 10\%$  of  $N$  nodes are chosen to behave maliciously (the trends reported in this chapter apply to all values of  $p$ ). We set the number of replicas of a file to be  $R = 7$  and vary the corruption threshold  $cr$  in our experiments. We simulated the bad nodes as having large but bounded power based on the parameters  $\alpha$  (DoS attack strength),  $\lambda$  (node compromise rate) and  $\mu$  (node recovery rate) (see the threat model in Section 2.3). We demonstrate the effectiveness of LocationGuard against DoS and host compromise based target file attacks.

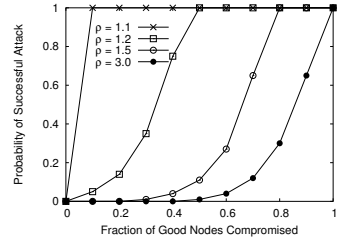
**Denial of Service Attacks.** Figure 13 shows the probability of an attack for varying  $\alpha$  and different values of corruption threshold ( $cr$ ). Without the knowledge of the location of file replicas an adversary is forced to attack (DoS) a random collection of nodes in the system and *hope* that that at least  $cr$  replicas of the target file is attacked. Observe that if the malicious nodes are more powerful (larger  $\alpha$ ) or if the corruption threshold  $cr$  is very low, then the probability of an attack is higher. If an adversary were aware of the  $R$  replica



**Figure 13:** Probability of a Target File Attack for  $N = 1024$  nodes and  $R = 7$  using DoS Attack



**Figure 14:** Probability of a Target File Attack for  $N = 1024$  nodes and  $R = 7$  using Host Compromise Attack (with no token collection)



**Figure 15:** Probability of a Target File Attack for  $N = 1024$  nodes and  $R = 7$  using Host Compromise Attack with token collection from compromised nodes

holders of a target file then a weak collection of  $B$  malicious nodes, such as  $B = 102$  (i.e., 10% of  $N$ ) with  $\alpha = \frac{R}{B} = \frac{7}{102} = 0.07$ , can easily attack the target file. Also, for a file system to handle the DoS attacks on a file with  $\alpha = 1$ , it would require a large number of replicas ( $R$  close to  $B$ ) to be maintained for each file. For example, in the case where  $B = 10\% \times N$  and  $N = 1024$ , the system needs to maintain as large as 100+ replicas for each file. Clearly, without LocationGuard, the effort required for an adversary to attack a target file is dependent only on  $R$ , but is independent of the number of good nodes ( $G$ ) in the system. On the contrary, LocationGuard based techniques scale the hardness of an attack with the number of good nodes in the system. Thus even with a very small  $R$ , a LocationGuard based system can make it very hard for any adversary to launch a targeted file attack.

**Host Compromise Attacks.** To further evaluate the effectiveness of LocationGuard against targeted file attacks, we evaluate LocationGuard against host compromise attacks. Our first experiment on host compromise attack shows the probability of an attack on the target file assuming that the adversary does not collect capabilities (tokens) stored at the compromised nodes. Hence, the target file is attacked if  $cr$  or more of its replicas are stored at either malicious nodes or compromised nodes. Figure 14 shows the probability of an attack for different values of corruption threshold ( $cr$ ) and varying  $\rho = \frac{\mu}{\lambda}$  (measured in number of node recoveries per node compromise). We ran the simulation for a duration

$\rho$	0.5	1.0	1.1	1.2	1.5	3.0
$G'$	0	0	0.05	0.44	0.77	0.96

**Table 5:** Mean Fraction of Good Nodes in Uncompromised State ( $G'$ )

of  $\frac{100}{\lambda}$  time units. Recall that  $\frac{1}{\lambda}$  denotes the mean time required for one malicious node to compromise a good node. Note that if the simulation were run for infinite time then the probability of attack is always one. This is because, at some point in time,  $cr$  or more replicas of a target file would be assigned to malicious nodes (or compromised nodes) in the system.

From Figure 14 we observe that when  $\rho \leq 1$ , the system is highly vulnerable since the node recovery rate is lower than the node compromise rate. Note that while a DoS attack could tolerate powerful malicious nodes ( $\alpha > 1$ ), the host compromise attack cannot tolerate the situation where the node compromise rate is higher than their recovery rate ( $\rho \leq 1$ ). This is primarily because of the cascading effect of host compromise attack. The larger the number of compromised nodes we have, the higher is the rate at which other good nodes are compromised (see the adversary model in Section 2.3). Table 5 shows the mean fraction of good nodes ( $G'$ ) that are in an uncompromised state for different values of  $\rho$ . Observe from Table 5 that when  $\rho = 1$ , most of the good nodes are in a compromised state.

As we have mentioned in Section 2.6.4, the adversary could collect the capabilities (tokens) of the file replicas stored at compromised nodes; these tokens can be used by the adversary at any point in future to corrupt these replicas using a simple write operation. Hence, our second experiment on host compromise attack measures the probability of a attack assuming that the adversary collects the file tokens stored at compromised nodes. Figure 15 shows the mean effort required to locate all the replicas of a target file ( $cr = R$ ). The effort required is expressed in terms of the fraction of good nodes that need to be compromised by the adversary to attack the target file.

Note that in the absence of LocationGuard, an adversary needs to compromise at most  $R$  good nodes in order to succeed a targeted file attack. Clearly, LocationGuard based techniques increase the required effort by several orders of magnitude. For instance, when  $\rho = 3$ , an adversary has to compromise 70% of the good nodes in the system in order to

$\rho$	0.5	1.0	1.1	1.2	1.5	3.0
Rekeying Interval	0	0	0.43	1.8	4.5	6.6

**Table 6:** Time Interval between Location ReKeying (normalized by  $\frac{1}{\lambda}$  time units)

$F$	4K	8K	16K	32K
$S_{max}$	12	13	14	15
$S_{zipf}$	7.75	8.36	8.95	9.55

**Table 7:** Entropy (in number of bits) of a Zipf-distribution

$\mu_{dep}$	0	1/256	1/16	1	16	256	4096	$\infty$
$S$	15	12.64	11.30	10.63	10.00	9.71	9.57	9.55

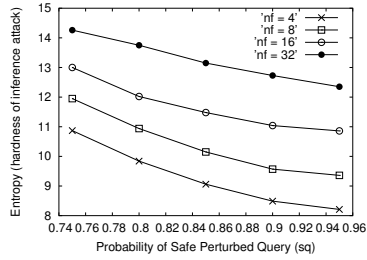
**Table 8:** Countering Lookup Frequency Inference Attack Approach I: Result Caching (with 32K files)

increase the probability of an attack to a nominal value of 0.1, even under the assumption that an adversary collects file capabilities from compromised nodes. Observe that if an adversary compromises every good node in the system once, it gets to know the tokens of all files stored on the overlay network. In Section 2.9.3 we had proposed location rekeying to protect the file system from such attacks. The exact period of location rekeying can be derived from Figure 15. For instance, when  $\rho = 3$ , if a user wants to retain the attack probability below 0.1, the time interval between rekeying should equal the amount of time it takes for an adversary to compromise 70% of the good nodes in the system. Table 6 shows the time taken (normalized by  $\frac{1}{\lambda}$ ) for an adversary to increase the attack probability on a target file to 0.1 for different values of  $\rho$ . Observe that as  $\rho$  increases, location rekeying can be more and more infrequent.

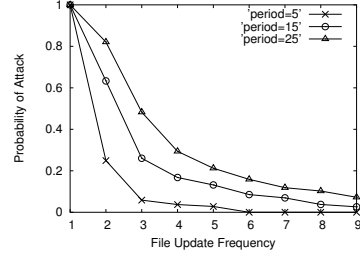
### 2.10.3 Location Inference Guards

In this section we show the effectiveness of location inference guards against the lookup frequency inference attack, and the file replica inference attack.

**Lookup Frequency Inference Guard.** We have presented lookup result caching and file identifier obfuscation as two techniques to thwart the frequency inference attack. Recall that our solutions attempt to flatten the frequency profile of files stored in the system (see Lemma 2.9.1). Note that we do not change the actual frequency profile of files; instead we flatten the apparent frequency profile of files as perceived by an adversary. We assume that files are accessed in proportion to their popularity. File popularities are derived from a Zipf-like distribution [119], wherein, the popularity of the  $i^{th}$  most popular file in the



**Figure 16:** Countering Lookup Frequency Inference Attack Approach II: File Identifier obfuscation



**Figure 17:** Countering File Replica Frequency Inference Attack: Location Rekeying Frequency Vs File Update Frequency

system is proportional to  $\frac{1}{v^\gamma}$  with  $\gamma = 1$ .

Our first experiment on inference attacks shows the effectiveness of lookup result caching in mitigating frequency analysis attack by measuring the *entropy* [59] of the apparent frequency profile (measured as number of bits of information). Given the apparent access frequencies of  $F$  files, namely,  $\lambda'_{f_1}, \lambda'_{f_2}, \dots, \lambda'_{f_F}$ , the entropy  $S$  is computed as follows. First the frequencies are normalized such that  $\sum_{i=1}^F \lambda'_{f_i} = 1$ . Then,  $S = -\sum_{i=1}^F \lambda'_{f_i} * \log_2 \lambda'_{f_i}$ . When all files are accessed uniformly and randomly, that is,  $\lambda'_{f_i} = \frac{1}{F}$  for  $1 \leq i \leq F$ , the entropy  $S$  is maximum  $S_{max} = \log_2 F$ . The entropy  $S$  decreases as the access profile becomes more and more skewed. Note that if  $S = \log_2 F$ , no matter how clever the adversary is, he/she cannot derive any useful information about the files stored at good nodes (from Lemma 2.9.1). Table 7 shows the maximum entropy ( $S_{max}$ ) and the entropy of a zipf-like distribution ( $S_{zipf}$ ) for different values of  $F$ . Note that every additional bit of entropy, doubles the effort required for a successful attack; hence, a frequency inference attack on a Zipf distributed 4K files is about 19 times ( $2^{12-7.75}$ ) easier than the ideal scenario where all files are uniformly and randomly accessed.

Table 8 shows the entropy of apparent file access frequency as perceived by an adversary when lookup result caching is employed by the system for  $F = 32K$  files. We assume that the actual access frequency profile of these files follows a Zipf distribution with the frequency of access to the most popular file ( $f_1$ ) normalized to one access per unit time. Table 8 shows the entropy of the apparent lookup frequency for different values of  $\mu_{dep}$  (the mean rate at

which a node joins/leaves the system). Observe if  $\mu_{dep}$  is large, the entropy of apparent file access frequency is quite close to that of Zipf-distribution (see Table 7 for 32K files); and if the nodes are more stable ( $\mu_{dep}$  is small), then the apparent frequency of all files would appear to be identically equal to  $\mu_{dep}$ .

In our second experiment, we show the effectiveness of file identifier obfuscation in mitigating frequency inference attack. Figure 16 shows the entropy of the apparent file access frequency for varying values of  $pr_{sq}$  (the probability that obfuscated queries are safe, see Theorem 2.8.1) for different values of  $nf$ , the mean number of files per node. Recall that an obfuscated identifier is safe if both the original identifier and the obfuscated identifier are assigned to the same node in the system. Higher the value  $pr_{sq}$ , smaller is the safe obfuscation range ( $srg$ ); and thus, the lookup queries for a replica location token are distributed over a smaller region in the identifier space. This decreases the entropy of the apparent file access frequency. Also, as the number of files stored at a node increases, there would be larger overlaps between the safe ranges of different files assigned to a node (see Figure 9). This evens out (partially) the differences between different apparent file access frequencies and thus, increases the entropy.

**File Replica Inference Guard.** We study the severity of file replica inference attack with respect to the update frequency of files in the file system. We measured the probability that an adversary may be able to successfully locate all the replicas of a target file using the file replica inference attack when all the replicas of a file are encrypted with the same key. We assume that the adversary performs a host compromise attack with  $\rho = 3$ . Figure 17 shows the probability of a successful attack on a target file for different values of its update frequency and different values of rekeying durations. Note that the time period at which location keys are changed and the time period between file updates are normalized by  $\frac{1}{\lambda}$  (mean time to compromise a good node). Observe the sharp knee in Figure 17; once the file update frequency increases beyond  $3\lambda$  (thrice the node compromise rate) then probability of a successful attack is very small.

Note that  $\lambda$ , the rate at which a node can be compromised by one malicious node is likely

to be quite small. Hence, even if a file is infrequently updated, it could survive a file replica inference attack. However, read-only files need to be encrypted with different cryptographic keys to make their replicas non-identical. Figure 17 also illustrates that lowering the time period between key changes lowers the attack probability significantly. This is because each time the location key of a file  $f$  is changed all the information collected by an adversary regarding  $f$  would be rendered entirely useless.

**Inference Attacks Discussion.** We have presented techniques to mitigate some popular inference attacks. There could be other inference attacks that have not been addressed in this chapter. Even the location inference guards presented in this chapter does not entirely rule out the possibility of an inference attack. For instance, even when we used result caching and file identifier perturbation in combination, we could not increase the entropy of apparent lookup frequency to the theoretical maximum ( $S_{max}$  in Table 7). Identifying other potential inference attacks and developing better defenses against the inference attacks that we have already pointed out in this chapter is a part of our ongoing work.

## 2.11 *Related Work*

Serverless distributed file systems like CFS [23], Farsite [7], OceanStore [53] and SiRiUS [39] have received significant attention from both the industry and the research community. These file systems store files on a large collection of untrusted nodes that form an overlay network. They use cryptographic techniques to secure files from malicious nodes. Unfortunately, cryptographic techniques cannot protect a file holder from DoS or host compromise attacks. LocationGuard presents low overhead and highly effective techniques to guard a distributed file system from such targeted file attacks.

Castro *et al.* [17] discuss several issues on secure routing in DHT based overlay networks. They suggest redundant routing as a solution for strengthening the routing scheme using a routing failure test based on the density of node identifiers. In redundant routing, the query source sends lookup query through different routes with a hope that at least one them eventually reaches the destination node. Note that our analysis on multiple near optimal independent paths is applicable to the case where redundant routing is used. Also note

that having multiple independent paths improves the probability of success and lowers the lookup cost for both repeated checking and redundant routing techniques.

The Sybil attack chapter [30] showed that entities (nodes) can forge multiple identities for malicious intent, thereby having a set of faulty entities represented through a larger set of identities. The chapter also suggests that one solution to the Sybil attack is using secure node IDs that are signed by well-known certifying agents. However, requiring every node to possess a certificate would turn out quite expensive; and may discourage even the good nodes from joining the system in the first place. Hence, one is forced to employ weak secure node IDs (that significantly reduce the number of identities owned by an entity). Therefore, this chapter (Section 2.5) assessed the risks and security threats when such weak secure IDs are deployed in a DHT-based overlay network.

The secure Overlay Services (SOS) chapter [51] presents an architecture that proactively prevents DoS attacks using secure overlay tunneling and routing via consistent hashing. However, the assumptions and the applications in [51] are noticeably different from that of ours. For example, the SOS chapter uses the overlay network for introducing randomness and anonymity into the SOS architecture to make it difficult for malicious nodes to attack target applications of interest. LocationGuard treats the overlay network as a part of the target applications we are interested in and introduce randomness and anonymity through location key based hashing and lookup based file identifier obfuscation, making it difficult for malicious nodes to target their attacks on a small subset of nodes in the system, who are the replica holders of the target file of interest.

The Hydra OS [21] proposed a capability-based file access control mechanism. LocationGuard implements a simple and efficient capability-based access control on a wide-area network file system. The most important challenge for LocationGuard is that of keeping a file's capability secret and yet being able to perform a lookup on it (see Section 2.8).

Indirect attacks such as attempts to compromise cryptographic keys from the system administrator or use fault attacks like RSA timing attacks, glitch attacks, hardware and software implementation bugs [65] have been the most popular techniques to attack cryptographic algorithms. Similarly, attackers might resort to inference attacks on LocationGuard

since a brute force attack (even with range sieving) on location keys is highly infeasible.

## ***2.12 Summary***

We have described LocationGuard – a technique for securing wide area serverless file sharing systems from targeted file attacks. Analogous to traditional cryptographic keys that hide the contents of a file, LocationGuard hides the location of a file on an overlay network. LocationGuard protects a target file from DoS attacks, host compromise attacks, and file location inference attacks by providing a simple and efficient access control mechanism with minimal performance and storage overhead. The unique characteristics of LocationGuard approach is the careful combination of location key, routing guard, and an extensible package of location inference guards, which makes it very hard for an adversary to infer the location of a target file by either actively or passively observing the overlay network. Our experimental results quantify the overhead of employing location guards and demonstrate the effectiveness of the LocationGuard scheme against DoS attacks, host compromise attacks and various location inference attacks.

Our research on LocationGuard continues along several dimensions. First, we are currently working on identifying other potential and possibly more sophisticated inference attacks and, aiming at developing better defenses against various inference attacks. Second, we are actively working on secure key distribution algorithms and investigating factors that can influence the frequency and timing of the rekeying process. Furthermore, we have started the development of LocationGuard toolkit as a value-added package that can be plugged on top of existing DHT-based P2P systems. We believe location hiding is an important property and that LocationGuard mechanisms are simple, secure, efficient and applicable to many large-scale overlay network based applications.

## CHAPTER III

### PUBLISH/SUBSCRIBE NETWORK SECURITY

Publish-subscribe (pub-sub) systems are an emerging paradigm for building large number of distributed systems. A wide area pub-sub system is usually implemented on an overlay network infrastructure that enables information dissemination from information publishers to subscribers. Using an open overlay network raises several security concerns such as: confidentiality & integrity, authentication, authorization and denial-of-service (DoS) attacks. In this chapter we present EventGuard – a dependable framework system architecture for securing wide area pub-sub systems. The EventGuard architecture comprises of two key components: (1) a suite of security guards that can be seamlessly plugged-into a content-based pub-sub system, and (2) a resilient pub-sub network design that is capable of scalable routing, handling message dropping-based DoS attacks and node failures. The design of EventGuard mechanisms aims at providing security guarantees while maintaining the system’s overall simplicity, scalability and performance metrics. We describe an implementation of the EventGuard pub-sub system to show that EventGuard is easily stackable on any content-based pub-sub core. We report two types of experiments performed to evaluate EventGuard. First, we use micro-benchmarks to quantify the overhead of EventGuard mechanisms and measure the performance and storage overheads. Then we use macro-benchmarks to quantify the overhead of the entire EventGuard pub-sub system and we also quantify the resilience of EventGuard to DoS attacks.

#### ***3.1 Introduction***

A growing number of emerging Internet applications requires information dissemination across different organizational boundaries, heterogeneous platforms, and a large, dynamic population of publishers and subscribers. A publish-subscribe overlay service is a wide-area communication infrastructure that enables data access and data sharing across potentially

unlimited number of publishers and subscribers, scattered geographically across the wired and wireless Internet [16]. A wide-area publish-subscribe (hereafter refer to as pub-sub) system is often implemented as a collection of spatially disparate computing nodes (network servers) communicating with each other through content-based routing protocols on top of a peer to peer overlay network [16]. In such an environment, publishers publish information in the form of event notifications and subscribers have the ability to express their interests in an event or a pattern of events by sending subscriptions to the pub-sub overlay network infrastructure. The pub-sub overlay network uses content-based routing schemes to dynamically match each publication to all the active subscriptions, and notifies the subscribers of any publication that matches their registered interest, ensuring that subscribers only receive notifications of those events that match their subscribed interests.

An important characteristic of pub-sub overlay services is the decoupling of publishers and subscribers combined with content-based routing protocols, enabling a many (publishers/subscribers) to many (subscribers/publishers) communication model. Such a model presents many inherent benefits as well as potential risks. On one hand, by offloading the task of identifying destination addresses of subscriptions (messages) from the publishers to the pub-sub overlay network, it not only allows message routing to be handled in a way that avoids unnecessary message replications but also enables dynamic and fine-grained subscriptions. As a result, the pub-sub overlay services have proven to be scalable and effective for wide-area information dissemination across distinct administrative domains and heterogeneous systems. On the other hand, many security concerns exist in such an environment regarding authenticity, confidentiality, integrity and availability of publications and subscriptions. For example, how can we guarantee that only the genuine publications are delivered to the subscribers (publication authenticity) and only the subscribers who subscribe (e.g., have paid) the service will receive the publications matching their interest (subscription authentication)? How do we prevent unauthorized modifications of pub-sub messages in transit (service integrity, publication and subscription integrity)? How do we perform content-based routing without publishers trusting the pub-sub network (content confidentiality)? Can subscribers receive publications without revealing their subscriptions

to the publishers (subscription confidentiality)? Can publishers enforce that only authorized subscribers receive particular publications (publication confidentiality)? And how do we defend the pub-sub services from publication spamming and flooding attacks, selective and random message dropping attacks, and other Denial of Service (DoS) or Denial of Information (DoI) attacks?

Not surprisingly, research and development of the pub-sub systems to date have been largely dedicated to the performance and scalability of pub-sub networks as well as the expressiveness of event publication and subscription models. Several distributed algorithms have been documented for efficient wide-area event matching and notification through a mesh of pub-sub overlay nodes [16, 12, 25]. Only recently, a few researchers have studied specific security requirements of pub-sub networks [109], pointing out attacks threatening message integrity (unauthorized writes) and authenticity (fake origins) in addition to message confidentiality (unauthorized reads), and the risks of bogus publications and fake subscriptions. Unfortunately, most of the existing secure event distribution protocols proposed so far focus only on the content confidentiality risks in pub-sub networks [76, 66]. Very few have devoted to developing a coherent security framework that can guard the pub-sub overlay services from multiple security problems inherent in the pub-sub overlay networks, such as decoupling publications from subscriptions while ensuring publication and subscription authenticity, content-aware routing while keeping the content from unauthorized reads and writes (confidentiality and integrity), subscription-based matching without compromising subscription confidentiality, and message forwarding while preventing message flooding and message dropping-based DoS and DoI attacks. The lack of these security guarantees has been a major hurdle in deploying pub-sub systems at large scale for mission-critical applications that could greatly benefit from them.

Pub-sub systems typically support two levels of event matching – *topic-based* and *content-based*. In a topic-based matching scheme [8], every event is marked with a topic. A topic could be a simple keyword or any unique numeric identifier. A subscriber subscribes to one or more topics, and receives all the events published under these topics. The pub-sub network routes events based on simple topic matching. Content-based matching schemes

[16, 9, 12] are layered on top of topic-based matching scheme and they allow more sophisticated event matching and filtering. For example, a subscriber may specify a *condition* on the event (say, `stock price > 100`) as a part of its subscription.

In this chapter, we present *EventGuard* – a dependable system architecture and a set of defense mechanisms for securing a pub-sub overlay service. EventGuard comprises of two key components. The first key component is a suite of security guards that are designed for protecting basic publish and subscribe operations from DoS attacks and unauthorized reads and writes. This component can be seamlessly plugged-into a wide-area content-based pub-sub system. The second key component is a resilient pub-sub network design that is capable of providing secure and yet scalable message routing, countering message dropping-based DoS attacks and node failures. A unique contribution of the EventGuard architecture is the development of a unified security framework that meets two important design objectives. On one hand, we aim at developing security mechanisms that are effective for guarding the pub-sub overlay services from various vulnerabilities and threats and ensuring authenticity, availability, confidentiality, and integrity of publications, subscriptions, and pub-sub overlay routing. On the other hand, we want to maintain the system’s overall simplicity, scalability and performance metrics while providing security guarantees. We also present a prototype implementation of EventGuard on top of Siena [16] to show that EventGuard is easily stackable on any content-based pub-sub core. With this prototype, we have conducted experimental evaluation of the overhead added by EventGuard to the pub-sub system by comparing EventGuard with Siena. Our experimental results show that EventGuard can secure a pub-sub network with minimal performance penalty.

The rest of this chapter is organized as follows. We first present a formal pub-sub system model and a threat model, which serve as the basic system model for the design of EventGuard in Section 3.2 and an overview of the EventGuard architecture in Section 3.3. Section 3.4 details the design of our security guards and Section 3.6 that describes EventGuard’s resilient network design. We present an implementation of EventGuard and evaluate it in Section 3.7. Section 3.8 discusses some related work followed by the conclusion in Section 3.9.

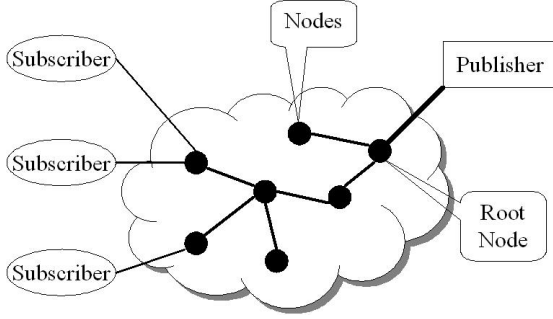


Figure 18: Basic Pub-Sub System

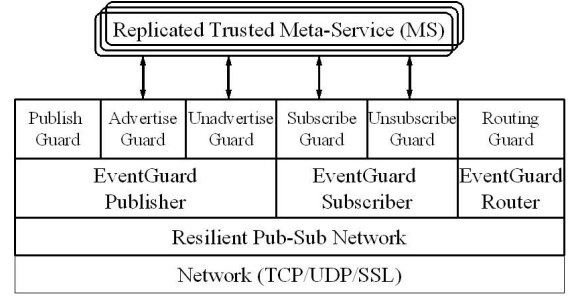


Figure 19: EventGuard Architecture

## 3.2 Preliminaries

### 3.2.1 Reference Pub-Sub Model

In content-based pub-sub systems, publishers publish their contents in terms of event notifications. An event notification is a set of attributes where an attribute is defined by its name, type, and value [16]. Subscribers have the ability to express their interest in an event by sending a subscription to the pub-sub overlay network infrastructure. The subscription is a predicate containing one or more constraints (filters). The infrastructure notifies the subscribers of any published notification that matches their subscribed interests. This section presents a reference pub-sub model. Our model is very similar to that used in a content-based pub-sub system like Siena [16]. Consider the stock quote dissemination where an example event notification consisting of the following attributes –  $\langle \text{exchange: string, NYSE} \rangle$ ,  $\langle \text{symbol: string, IBM} \rangle$ ,  $\langle \text{price: float, 122} \rangle$  (the latest value of the specified stock symbol),  $\langle \text{volume: integer, 2500} \rangle$  (the latest volume of transactions in thousands). An example subscription could be  $\langle \text{symbol} = \text{'IBM'}, \text{exchange} = \text{'NYSE'}, \text{price} > 100 \rangle$ .

A typical pub-sub system implements five important primitives: *subscribe*, *advertise*, *publish*, *unsubscribe* and *unadvertise*. Subscribers specify the events in which they are interested using the *subscribe* function. Publishers advertise the type of events they would publish using *advertise*. Publishers publish events via the *publish* function. A subscription is repeatedly matched until it is canceled by a call to *unsubscribe*. An advertisement remains in effect until it is canceled by an *unadvertise*. We use the term messages to loosely denote

all traffic on the pub-sub network, including publications, subscriptions, advertisements, unsubscriptions and unadvertisements.

Publications are specified in terms of events and subscriptions are expressed in terms of predicate filters. Formally, an event  $e = \langle \alpha \rangle^* = \langle name, type, value \rangle^*$ , where  $\alpha$  is some attribute of the form  $\langle name, type, value \rangle$ ,  $name$  refers to some attribute name,  $type$  refers to the data type of the attribute,  $value$  corresponds to its published value, and the notation  $*$  indicates that an event may comprise of one or more attributes. A *filter* selects events by specifying a set of attributes and constraints on the values of those attributes. Formally, filter  $f = \langle \phi \rangle^* = \langle name, operator, value \rangle^*$ , where  $\phi$  is some constraint of the form  $\langle name, operator, value \rangle$ ,  $name$  refers to some attribute name,  $value$  specifies an attribute value,  $operator$  refers to a binary operator, and the notation  $*$  indicates that a filter may comprise of one or more constraints in a conjunctive form. Operators typically include common equality and ordering relations ( $=$ ,  $<$ ,  $>$ , etc) for numeric types; and substring, prefix, suffix operators for strings.

An attribute  $\alpha = \langle name_\alpha, type_\alpha, value_\alpha \rangle$  satisfies a constraint  $\phi = \langle name_\phi, operator_\phi, value_\phi \rangle$  if and only if  $name_\alpha = name_\phi$ ,  $value_\phi$  is of  $type_\alpha$ , and  $operator_\phi(value_\alpha, value_\phi)$  is true. When an attribute  $\alpha$  satisfies a constraint  $\phi$ , we say that  $\alpha$  *matches*  $\phi$ . Equivalently, when  $\alpha$  matches  $\phi$ , we say that  $\phi$  *covers*  $\alpha$ . For example, an attribute  $\alpha = \langle price, 120 \rangle$  matches the constraint  $\phi = \langle price, \geq, 100 \rangle$ . An event  $e$  matches a subscription filter  $f$  if for all constraints  $\phi$  in  $f$ , there exists some attribute  $\alpha$  in  $e$  that matches  $\phi$ . When a filter is used in an advertisement, it defines the set of all possible notifications that can be generated by the publisher. An event  $e$  matches an advertisement filter  $f$  if for all attributes  $\alpha$  in  $e$ , there exists some constraint  $\phi$  in  $f$  that covers  $\alpha$ . The notion of *covers* can be extended in a straightforward manner to two subscription filters, or two advertisement filters or a subscription filter and an advertisement filter.

Unsubscriptions and unadvertisements serve to cancel previous subscriptions and advertisements respectively. Given an unsubscription  $unsubscribe(X, f)$ , where  $X$  is the identity of the subscriber and  $f$  is a filter, the pub-sub system cancels all subscriptions  $subscribe(X, g)$  submitted by the subscriber  $X$  with subscription filter  $g$  covered by  $f$ . Similarly, an

unadvertisement message  $unadvertise(Y, f)$  cancels all advertisements  $advertise(Y, g)$  submitted by the publisher  $Y$  with advertisement filter  $g$  covered by  $f$ .

As illustrated in Figure 18, in a wide-area pub-sub system, publishers and subscribers are usually outside the pub-sub network (though not required). Typically, we have a relatively small set of known and trusted publishers and a much larger set of unknown subscribers. A natural choice for the topology of a pub-sub network is a hierarchical topology (see Figure 18). Other plausible topologies include peer-to-peer and mixed topologies like super-peer topology [16]. For the sake of simplicity, in this chapter we assume a hierarchical topology for the pub-sub network. When a node  $n$  receives a subscription request  $subscribe(m, f)$  from node  $m$ , it registers filter  $f$  with identity  $m$ . If filter  $f$  is not covered by any previously subscribed filters at node  $n$  then node  $n$  forwards  $subscribe(n, f)$  to its parent node. Note that node  $m$  could be a subscriber or simply another node in the pub-sub overlay network that forwarded a subscription request to node  $n$ .

Effectively, for every pair of publisher and topic (publication), a pub-sub dissemination tree is constructed with the publisher as the root, the subscribers as the leaves and the pub-sub routing nodes as the intermediate nodes of the tree. The publications flow from the root (publisher) to the leaves (subscribers) of the tree. Similarly, advertisements, unsubscriptions and unadvertisements are propagated from the root to the leaves of the tree. Note that a node  $n$  in the pub-sub network may belong to one or more pub-sub dissemination trees (or so called pub-sub network channels), and each corresponds to a publisher and a topic of events that the publisher publishes through this channel. When a node  $n$  receives a publication notification  $publish(Y, e)$  from  $Y$  to publish the event  $e$ , it uses the pub-sub dissemination tree to which it belongs to identify all active subscriptions whose filters  $\{f_1, f_2, \dots, f_p\}$  are matched by the event  $e$ . Then, node  $n$  identifies and forwards event  $e$  to those of its children nodes  $\{X_1, X_2, \dots, X_q\}$  that have subscriptions with subscription filters covered (matched) by a subset of filter  $f_i$  ( $1 \leq i \leq p$ ).

### 3.2.2 Threat Model

The pub-sub overlay service model comprises of three entities: publishers, subscribers and routing nodes. In this section, we present our threat model for all these entities.

**Publishers.** EventGuard assumes that authorized publishers are honest. All publications by authorized publishers are assumed to be valid and correct. However, one could build a feedback mechanism wherein the subscribers rate the publishers periodically [119]. Over a period of time, subscribers would subscribe only to high quality publishers and the low quality publishers would eventually run out of business. However, unauthorized publishers may *masquerade* as an authorized publishers and *flood* the network and consequently the subscribers, with incorrect or duplicate publications, advertisements or unadvertisements.

**Subscribers.** EventGuard assumes that authorized subscribers may be *partially dishonest*. Concretely, we assume that an authorized subscriber does not reveal publications to other unauthorized subscribers (otherwise, this would be equivalent to solving the digital copyrights problem). However, *unauthorized subscribers* may be curious to obtain information about publications to which they have not subscribed. Also, subscribers may attempt to *spam* or *flood* the pub-sub network with duplicate or fake subscriptions and unsubscriptions.

**Routing nodes.** EventGuard assumes that some of the nodes on the pub-sub network may exhibit dishonest behavior. However, we also assume that a significant fraction of the pub-sub nodes are non-malicious so as to ensure that the network is *alive*. A pub-sub network is alive if it can route messages and maintaining its connectivity despite the presence of malicious nodes. Malicious nodes may *eavesdrop* or *corrupt* pub-sub messages routed through them. Malicious nodes may also attempt to selectively or randomly drop pub-sub messages. Further, malicious nodes may attempt to *flood* other nodes and subscribers.

Finally, EventGuard assumes that the underlying IP-network may be not guarantee confidentiality, integrity or authenticity. However, we assume that the underlying domain name service (DNS), the network routers, and the related networking infrastructure is completely secure, and hence cannot be subverted by a malicious node. Due to space

constraints, in this chapter we describe EventGuard mechanisms in the context of a topic-based pub-sub system. However, EventGuard mechanisms can be extended to handle more complex event filtering techniques that use sophisticated filtering conditions on numeric attributes and subject (concept) hierarchies.

### ***3.3 EventGuard Overview***

#### **3.3.1 Design Goals**

EventGuard has fundamentally two sets of design goals: security goals and performance goals. In EventGuard, providing a safeguard mechanism that defines who has what access to which information is considered an application-level protection issue and involves publishers and subscribers. It requires a definition of identity, authorization, and access control within the pub-sub overlay network. However, controlling who is able to change the subscription database maintained by the pub-sub service and to restrict channel utilization is mostly an overlay network service level concern. We argue that it is impractical for the pub-sub system to impose a global security policy as the one-size-fit-all solution to securely disseminate information for every application. A major challenge in designing a secure pub-sub service framework is the flexibility to allow diverse policies and mechanisms to co-exist within the same pub-sub system. We below describe EventGuard’s security goals and performance goals at both application level and overlay network level:

**Authentication.** In a pub-sub system a publication (or an advertisement) is sent from a publisher (sender) to a subscriber (receiver) through the pub-sub network (channel). It is important to make sure that all publications (and advertisements) are authentic in order to avoid spoofed publication and spam. On the other hand, subscription authenticity is important when the application requires that subscribers should receive only the publications to which they are authorized (paid) to access. In addition, sender authentication within the pub-sub content routing network is critical when malicious fault arise at the pub-sub network level (e.g., hosts on the network are compromised). A malicious node can insert bogus subscriptions, or ignore the routing algorithm and route messages to arbitrary destinations.

**Confidentiality and Integrity.** Confidentiality and integrity of a message sent from

party  $A$  to party  $B$  is defined with respect to the channel connecting  $A$  and  $B$ . In a pub-sub system the concerned parties are the publishers and the subscribers, and the channel denotes the entire pub-sub service network. We require that the pub-sub network nodes or any observer of the pub-sub network should neither be able to gain knowledge about the messages routed through them nor corrupt them in an undetectable manner. Concretely, we need to guarantee three types of confidentiality and integrity.

- First, we need publication confidentiality to ensure only authorized subscribers can read an event and content must be protected from unauthorized subscribers and malicious users on the pub-sub network. We also need publication integrity to protect publications from unauthorized modification in transit.
- Second, subscribers may wish to keep their subscriptions private. Concretely, the subscriber would like the pub-sub network to compute the subscription function (filter)  $f(x)$  with respect to the publication  $x$  without revealing  $f$  to the network. We refer to this goal as subscription confidentiality, which ensures subscribers to receive content sensitive information without revealing their subscriptions to the publishers or the pub-sub overlay service network. Further, we need subscription integrity to safeguard subscriptions from unauthorized modification when routing them using the pub-sub overlay.
- Third, we need the pub-sub network to perform content-based routing without requiring publishers and subscribers to trust the network with the content. Content confidentiality is especially important when information being published contains sensitive content, thus publishers and subscribers may wish to keep information secret from the pub-sub service network. Content integrity prevents messages in transit from unauthorized access and modification. Authorization is a common technique for preventing unauthorized reads and writes of messages by illegal subscribers or curious (semi-honest) routing nodes in the pub-sub network.

**Availability.** EventGuard refers availability to the resilience of the pub-sub system against Denial of Service (DoS) attacks. There are three major types of DoS attacks possible on

pub-sub systems: (i) *flooding* based attacks attempt to flood the pub-sub system with large amount of bogus messages, (ii) *fake* unsubscribe (and unadvertise) attack attempts to send spurious unsubscribe (and unadvertise) requests; for example, if a node  $X'$  ( $\neq X$ ) sends  $unsubscribe(X, f)$  to node  $X$ 's parent then it would deny  $X$  of all events  $e$  that is covered by filter  $f$ , and (iii) *selective and random dropping* attack attempts to drop messages either selectively (say, based on the publication's topic) or randomly.

In addition to the security goals, EventGuard has two important performance related goals. *Performance and Scalability*: We require the EventGuard mechanisms to *scale* with the number of nodes in the network. In addition, EventGuard should add minimal performance overhead to a pub-sub system. *Ease of Use and Simplicity*: We require that EventGuard mechanisms be simple and easy to deploy, operate and administer.

### 3.3.2 System Architecture

EventGuard is designed to be completely modular and operates entirely above a content-based pub-sub *core*. EventGuard requires minimal coupling with the pub-sub core and hence can be easily ported from one pub-sub core to another. Figure 19 shows EventGuard's architecture. EventGuard comprises of three components. The first component is a suite of security guards that guard the pub-sub system from various security threats discussed in Section 3.2.2 and Section 3.3.1. The second component is a resilient pub-sub network that is capable of handling node failures and selective and random dropping-based DoS attacks. The third component is a light-weight trusted meta-service (MS) to provide identification and authorization control for advertisements and subscriptions in the pub-sub system.

**Security Guards.** EventGuard takes a unified approach to secure a pub-sub network. EventGuard's security guards handle one pub-sub operation against all potential attacks. Concretely, EventGuard comprises of six guards, securing six critical pub-sub operations: subscribe guard, advertise guard, publish guard, unsubscribe guard, unadvertise guard and routing guard. These guards are built on top of three important building blocks: token, key and signature. We use tokens as pseudo-names for publishing events on certain topics to mitigate selective dropping attacks at the pub-sub network level. We protect confidentiality

and integrity from pub-sub nodes and from unauthorized subscribers using cryptographic keys. We prevent the pub-sub service from spam, flooding based DoS attacks and spoofed messages using signatures. We design and develop the six security guards by composing these three building blocks in different ways to safeguard publication, subscription, unsubscription, advertisement and unadvertisement operations in the pub-sub system. We describe the three basic building blocks in Section 3.4.1 and discuss how to design security guards using these building blocks in the rest of Section 3.4.

**Resilient pub-sub network.** EventGuard achieves resilience to node failures and message dropping based attacks by constructing a network topology that is richer than the popular tree-based event dissemination topology. Although a strict tree-based topology minimizes the communication cost in the pub-sub content routing network, it is not robust for handling node failures and message dropping attacks [96]. We improve the resilience of the pub-sub network by modifying the tree structure to incorporate multiple independent paths [97] from the publisher to subscribers.

**Trusted Meta-Service.** EventGuard relies on a thin *trusted* meta-service (*MS*) to create tokens and keys for controlling confidentiality of topics (publications), and create signatures for authenticating messages, maintaining message integrity and guarding the pub-sub service from flooding-based DoS attacks. Four of the six types of pub-sub messages, such as subscribe, unsubscribe, advertise, unadvertised, require *MS* to generate tokens, keys and signatures. However, the most common operations, publish and routing, do not require the direct support from *MS*. We design the *MS* with the following three objectives in mind. First, we aim at minimizing the amount of work assigned to the *MS*. Keeping the *MS* simple enables one to ensure that the *MS* is relatively bug-free and thus is well-protected from malicious nodes. Second, we would like to limit the number of secrets maintained by the *MS* to at most one or two small strings (keys). Having to maintain only a few small keys secret enables the *MS*'s administrator to afford physical security for those keys in the form of, say a smart card. Third, it should be possible to easily replicate *MS*; the *MS* replicas should be able to function independently without having to interact with one another.

There are other potential benefits of supporting a light-weight *MS* in the pub-sub systems. For instance, if the pub-sub system wants to impose a cost model on the pub-sub system to ensure that subscribers pay the system for their subscriptions and publishers pay the system for their advertisements (*accounting*), the *MS* should be capable of accounting and auditing important control activities on the pub-sub network. Accounting and pricing can be valuable means to reduce spam in wide-area distributed systems. One example is to condone email spam by associating a cost with every email [20]. Furthermore, if the application wishes to have the pub-sub system maintain an *audit* trail of subscriptions, unsubscriptions, advertisements and unadvertisements, one can also provide auditing capability at the *MS* to resolve any accounting or pricing related issues regarding subscribers and publishers.

### ***3.4 EventGuard: Security Guards***

In this section we present a high level functional overview of EventGuard. We first introduce the three building blocks used by EventGuard: tokens, keys and signatures. Then we describe how EventGuard uses these primitives to develop six safeguards for securing the six important pub-sub operations: subscribe, advertise, publish, unsubscribe, unadvertise and routing.

#### **3.4.1 Tokens, Keys and Signatures**

The first building block is the concept of per-topic token. In a topic In EventGuard, publishers can publish and advertise events in terms of topics. We create a token for each topic. Tokens are essential for protecting messages (e.g., subscriptions) from selective dropping DoS attack. Concretely, by introducing tokens, nodes in the pub-sub network are not aware of the topic names of concrete events (publications); instead they match and route events on the pub-sub network based on tokens until the publications (event notifications) ultimately reach the appropriate subscribers.

Token is a pseudonym for a topic name. There is a one-to-one mapping between a topic name  $w$  and it's token  $T(w)$ . However, given a token  $T(w)$  it is computationally infeasible

to guess the topic name  $w$ . A subscriber subscribes for a topic  $w$  by subscribing for its token with filter  $f(w) = \langle \mathbf{topic}, EQ, T(w) \rangle$ , where  $\mathbf{topic}$  is the attribute name for topic,  $EQ$  denotes the equality operator, and  $T(w)$  denotes the token corresponding to topic  $w$ . The nodes in the pub-sub network are not aware of the topic names; instead they route events on the pub-sub network based on tokens. The nodes match and route events based on tokens until they ultimately reach the appropriate subscribers. Tokens serve are useful for achieving confidentiality and protection from selective dropping DoS attack. Per topic tokens are required for us to retain scalability by content-aware routing and yet achieve our security goals.

The second building block is the concept of per-topic key. In an EventGuard powered pub-sub system, keys are fundamental for achieving confidentiality and integrity. By encrypting message content with a random encryption key, we can prevent contents of messages (e.g., publications, subscriptions) from unauthorized reads and writes. Every topic  $w$  in the pub-sub system has an associated key  $K(w)$ . The  $MS$  is responsible for providing  $K(w)$  to a subscriber when the subscriber subscribes for topic  $w$  and to publishers when they advertise for topic  $w$ . The encryption key  $K(w)$  enables the publisher to encrypt events that belong to topic  $w$ . Now only a legal subscriber to topic  $w$  would be able to decrypt the message. The publication of content  $pbl$  under a topic  $w$  would be constructed as  $e = \langle \alpha_1, \alpha_2 \rangle$ , where  $\alpha_1 = \langle \mathbf{topic}, T(w) \rangle$  and  $\alpha_2 = \langle \mathbf{content}, E_{K(w)}(pbl) \rangle$ , where  $\mathbf{content}$  denotes the attribute name for the published message. Note that  $E_K(pbl)$  denotes the encryption of  $pbl$  with encryption key  $K$  and some symmetric key encryption algorithm  $E$  (e.g., DES [36] or AES [63]). The nodes in the pub-sub system are not aware of keys. The nodes would be able to route the event based on token  $T(w)$ . Keys are fundamental for achieving authorization, confidentiality and integrity. Per topic keys are required to guarantee fine grained authorization.

The third building block in EventGuard is the concept of signature. Signatures play a fundamental role in achieving message authentication and protecting the pub-sub services from flooding-based DoS attacks. EventGuard uses a probabilistic signature algorithm for achieving authenticity. A signature scheme is probabilistic if there are many possible

valid signatures for each message and the verification algorithm accepts any of the valid signatures as authentic. In the first prototype of EventGuard, we use ElGamal [34] as the probabilistic signature algorithm. A signature on any message  $M$  using ElGamal yields a tuple  $\langle r, s \rangle$ . The  $r$ -component of the signature is guaranteed to be unique (with high probability). Further, if the same message  $M$  is signed twice by the same entity  $x$ , we get two different, but valid ElGamal signatures of  $M$ . All messages originating at entity  $x$  are signed using its private key  $rk(x)$ ; and all its signatures are verified using its corresponding public key  $pk(x)$ . EventGuard uses the trusted meta-service  $MS$  to create signatures for advertisements and subscriptions. Subscriptions and advertisements are authenticated using signatures, ensuring that malicious nodes cannot flood the pub-sub network with bogus publications or phony subscriptions.

There are at least two alternative approaches to signatures. One apparent alternative is to use keyed message authentication codes (MACs). Shared MAC keys between a publisher and a subscriber allow the subscriber to authenticate all publications it receives. There is a dilemma with this approach. On one hand, we cannot afford to give away MAC keys to pub-sub network nodes since a malicious node may flood publications on the pub-sub network. On the other hand, without these MAC keys, nodes on the pub-sub network could neither verify the authenticity of messages nor control flooding based DoS attacks.

The second alternative to signatures is to use a Byzantine fault-tolerant (BFT) information dissemination protocol [57]. Let  $m$  denote an upper bound on the number of malicious nodes in the pub-sub network. The publisher initiates a publication message  $P$  by sending it to  $2m + 1$  seed nodes. Any non-seed node  $u$  would consider the message  $P$  authentic if and only if it received  $m + 1$  identical copies of the message  $P$  from other nodes in the system. Note that if  $m + 1$  copies of a message are identical, then at least one of the copy is guaranteed to have originated from a non-malicious node. Node  $u$  continues propagating the message  $P$  (usually by broadcast) until all subscribers receive the message  $P$ . However, this solution does not directly address confidentiality and integrity. Techniques similar to the ones we propose in EventGuard can be used to handle them. An obvious advantage of BFT techniques is that BFT does not pay the overhead of using a PKI based signature.

On the flip side however, BFT techniques incur much higher communication cost. This makes BFT techniques suitable only for environments that inherently support broadcast communication (e.g., local area networks) and for scenarios where information dissemination is required to be absolutely lossless. For wide-area Internet applications like pub-sub systems wherein one can tolerate lossy channels, it is important to keep the communication cost very low.

We have introduced tokens, keys and signatures as fundamental building blocks of EventGuard. The next challenge is to design and construct the six concrete safeguards for the following six essential operations: subscribe, advertise, publish, unsubscribe, unadvertise and routing.

### 3.4.2 Subscribe Guard

Subscribe guard is designed for achieving subscription authentication, subscription confidentiality and subscription integrity, and preventing DoS attacks based on spurious subscriptions. Suppose that a subscriber  $S$  wishes to subscribe for a topic  $w$ . In EventGuard, subscriber  $S$  sends the topic  $w$  to the EventGuard trusted meta service  $MS$  indicating that it wishes to subscribe for topic  $w$ . At this point, the  $MS$  may act as the authority for implementing a cost model for the pub-sub system. For example, the  $MS$  may collect a subscription fee for every subscription; the subscription fee may be dependent on the topic  $w$ . Let  $\phi'(w)$  be the original subscription filter for topic  $w$  sent to  $MS$  by the subscriber  $S$ ,  $sb(w)$  denote the subscription permit issued by  $MS$  upon receiving an subscription  $\phi'(w)$  from subscriber  $S$ , and  $\phi(w)$  denote the legal subscription transformed from  $\phi'(w)$  by  $MS$  in two steps: (1) replacing topic  $w$  with token  $T(w)$  and (2) signing the subscription with the subscription signature provided by  $MS$ . Both are included in the subscription permit  $sb(w)$  generated by  $MS$ . They are defined as follows:

$$\begin{aligned}\phi'(w) &= \langle \text{topic}, EQ, w \rangle \\ sb(w) &= \langle K(w), T(w), sig_{MS}^S(T(w)), UST^S(w) \rangle \\ \phi(w) &= \langle \text{topic}, EQ, T(w_i) \rangle, \langle sig, ANY, sig_{MS}^S(T(w_i)) \rangle\end{aligned}$$

The  $MS$  sends a subscription permit  $sb(w)$  to the subscriber  $S$ . The key  $K(w)$  for topic  $w$  is derived as  $K(w) = KH_{rk(MS)}(w)$ , where  $rk(MS)$  denotes the  $MS$ 's private key and  $KH_K(w)$  denotes a keyed hash of string  $w$  using a keyed-hash function  $KH$  (say HMAC-MD5 [52]) and a secret key  $K$ . The token  $T(w)$  for topic  $w$  is derived as  $T(w) = H(K(w))$ , where  $H(x)$  denotes a hash of string  $x$  using a one-way hash function  $H$  (say, MD5 [80] or SHA1 [32]).  $UST^S(w)$  is a special token given to the subscriber to enable safe unsubscription (discussed later under unsubscribe guard). Observe that if any two subscribers subscribe for topic  $w$ , they get the same encryption key  $K(w)$  and the same token  $T(w)$ .

The signature  $sig_{MS}^S(T(w))$  is an ElGamal signature by the  $MS$  on the token  $T(w)$  in the subscription permit  $sb(w)$  provided to subscriber  $S$ . The signature has two parts  $sig_{MS}^S(T(w)) = \langle r, s \rangle$ . Note that the  $r$ -component of the signature is always unique. Therefore, we use  $r$ -component of the signature as the subscription identifier. This signature serves us three purposes. First, it enables nodes in the pub-sub network to check the validity of a subscription. Second, we use the subscription identifier (the  $r$ -component of the signature) to detect and condone DoS attacks based on subscription flooding. Note that even if two subscribers  $S$  and  $S'$  subscribe for the same topic  $w$ ,  $sig_{MS}^S(T(w)) \neq sig_{MS}^{S'}(T(w))$  (discussed later under routing guard). Third, it is used to construct the special token  $UST^S(w) = KH_{rk(MS)}(r)$  where  $r$  denotes the  $r$ -component of the  $MS$ 's signature. We use  $UST^S(w)$  to prevent DoS attacks based on fake unsubscription (discussed later under unsubscribe guard).

Upon receiving a subscription permit  $sb(w)$  from the  $MS$ , subscriber  $S$  transforms its original subscription filter  $\phi'(w)$  to a legal subscription filter  $\phi(w)$  by signing the subscription using token  $T(w)$  and subscription signature  $sig_{MS}^S(T(w))$ . The subscriber  $S$  could then submit and deploy the signed subscription on the pub-sub network. Consequently, any publication that includes the token  $T(w)$  is routed to  $S$ . Routing nodes on the pub-sub network are not able to perform unauthorized reads or writes on the content of any subscription message, thus guaranteeing subscription confidentiality and integrity. Further, nodes compromised due to DoS attacks, even though malicious, are not able to attack the pub-sub network by flooding fake subscriptions.

A subscriber  $S$  may restrict the number of publications it would like to receive. For example, a subscriber may use  $sb(w_1)$  and  $sb(w_2)$  to construct a subscription filter that is a conjunction of filters  $f(w_1)$  and  $f(w_2)$ . In general a subscription filter  $f = \langle \phi(w_1), \phi(w_2), \dots, \phi(w_m) \rangle$ , where  $\phi(w)$  described above.

### 3.4.3 Publish Guard

Publish guard is designed to safeguard the publication from publication confidentiality and integrity, publication authenticity, and DoS attacks based on bogus publications and spam. Suppose a publisher  $P$  wishes to publish a publication  $pbl$  under topics  $w_1, w_2, \dots, w_m$ . The topics are used to categorize the content  $pbl$ . The content  $pbl$  could be any arbitrary sequence of bytes including text, multimedia, and so on. For each topic  $w_i$ , the publisher fetches the topic's token  $T(w_i)$  and its encryption key  $K(w_i)$  from the  $MS$ . A publication event  $e$  is constructed as follows. Let  $e'$  denote the original publication message,  $e$  denote a legal event publication transformed from  $e'$  using tokens and content encryption of publication messages. We formally define them as follows:

$$\begin{aligned}
 e' &= \langle \langle \text{publisher}, P \rangle, \langle \text{content}, pbl \rangle, \langle \text{topic}, w_1 \rangle, \dots, \langle \text{topic}, w_m \rangle \rangle \\
 e &= \langle \langle \text{publisher}, P \rangle, \langle \text{content}, E_{K_r}(pbl) \rangle, \langle \text{topic}, T(w_1) \rangle, \langle T(w_1), E_{K(w_1)}(K_r) \rangle, \dots, \\
 &\quad \langle \text{topic}, T(w_m) \rangle, \langle T(w_m), E_{K(w_m)}(K_r) \rangle \rangle
 \end{aligned}$$

The key  $K_r$  is a random encryption key generated each time a publisher needs to publish an event.  $P$  sends the event  $e$  along with its signature, namely,  $sig_P(e)$ . Observe that any subscriber for topic  $w_i$  possesses the key  $K(w_i)$ . An authorized subscriber uses the key  $K(w_i)$  to decrypt the random key  $K_r$ , and uses the random key  $K_r$  to decrypt the publication  $pbl$ .

Note that a publisher uses an ElGamal signature to sign its publications. The first component of the signature is used as the publication identifier. The signature serves two purposes. First, it enables nodes in the pub-sub network to check the validity of a publication. Second, we use the publication identifier (the  $r$ -component of the signature) to detect and condone a DoS attack based on publication flooding (discussed later under

routing guard).

#### 3.4.4 Advertise Guard

Advertise guard is designed for achieving advertisement authentication, advertisement confidentiality and integrity, and preventing DoS attacks based on bogus advertisement. Suppose a publisher  $P$  wishes to publish events under topic  $w$ . Publisher  $P$  sends  $w$  and its public-key  $pk(P)$  to the  $MS$ . At this point the  $MS$  may charge a publication fee to the publisher that is some arbitrary function of  $w$ .  $\phi'(w)$  is the original advertisement filter for topic  $w$ .

$$\begin{aligned}\phi'(w) &= \langle \text{publisher}, EQ, P \rangle, \langle \text{topic}, EQ, w \rangle \\ ad(w) &= \langle K(w), T(w), sig_{MS}^P(T(w) \parallel P \parallel pk(P)), UAT^P(w) \rangle \\ \phi(w) &= \langle \text{publisher}, EQ, P \rangle, \langle \text{topic}, EQ, T(w) \rangle, \langle sig, ANY, sig_{MS}^P(T(w) \parallel P \parallel pk(P)) \rangle\end{aligned}$$

The  $MS$  sends an advertisement permit  $ad(w)$  to the publisher  $P$ . The key  $K(w)$ , the token  $T(w)$  and the signature  $sig_{MS}^P(T(w) \parallel P \parallel pk(P))$  is computed in the same manner as that for subscriptions. The special token  $UAT^P(w)$  is used for unadvertisements (discussed in unadvertise). The publisher then constructs the advertisement filter  $\phi$  and propagates it to the pub-sub network. Note that the public-key  $pk(P)$  is essential for the pub-sub nodes and the subscribers to verify the authenticity of publications.

#### 3.4.5 Unsubscribe Guard

Unsubscribe guard is designed to prevent unauthorized unsubscribe messages, flooding of unsubscribe messages, and spam. When a subscriber  $S$  wishes to unsubscribe from a topic  $w$ ,  $S$  sends  $\langle T(w), sig_{MS}^S(T(w)), UST^S(w) \rangle$  to the  $MS$ . The  $MS$  checks if  $sig_{MS}^S(T(w))$  is a valid signature on  $T(w)$ . The  $MS$  uses the special token  $UST^S(w)$  to ensure protection from DoS attacks based on fake unsubscription. The  $MS$  checks if  $UST^S(w)$  is indeed equal to  $KH_{rk(MS)}(sbId)$ , where  $sbId$  denotes the subscription identifier, namely, the  $r$ -component of the signature  $sig_{MS}^S(T(w))$ . Note that the signature  $sig_{MS}^S(T(w))$  and the token  $T(w)$  are sent to the pub-sub network nodes when the subscriber  $S$  subscribes for the topic  $w$ . However, the subscriber  $S$  is never required to reveal the special token  $UST^S(w)$

to the pub-sub network. Hence, no malicious node in the pub-sub network would be able to fake an unsubscribe request. Moreover, the use of  $UST^S(w)$  prevents some subscriber  $S' (\neq S)$  who has subscribed for topic  $w$  (and thus possesses signature  $sig_{MS}^{S'}(T(w))$ , token  $T(w)$  and key  $K(w)$ ) from unsubscribing subscriber  $S$  from topic  $w$ . We use  $\phi'(w)$  to denote the original unsubscription message for topic  $w$ .

$$\begin{aligned}\phi'(w) &= \langle \mathbf{topic}, EQ, w \rangle \\ usb(w) &= \langle sig_{MS}(T(w) \parallel sbId) \rangle \\ \phi(w) &= \langle \mathbf{topic}, EQ, T(w) \rangle, \langle sig, ANY, sig_{MS}(T(w) \parallel sbId) \rangle\end{aligned}$$

The  $MS$  sends an unsubscription permit  $usb(w)$  to the subscriber  $S$ . Note that the signature includes the token  $T(w)$  and the original subscription's identifier  $sbId$ . Subscriber  $S$  would unsubscribe from topic  $w$  by sending  $\phi(w)$  to the pub-sub network. Nodes in the network use the  $MS$ 's signature to check the validity of an unsubscription and use the unsubscription identifier  $usbId$  (the  $r$  component of signature  $sig_{MS}(T(w) \parallel sbId)$ ) to detect and condone DoS attacks based on unsubscription flooding.

### 3.4.6 Unadvertise Guard

Unadvertised guard is designed to prevent the pub-sub network from unadvertisement flooding. When a publisher  $P$  wishes to unadvertise for a topic  $w$ ,  $P$  sends  $\langle T(w), sig_{MS}^P(T(w) \parallel P \parallel adId), UAT^P(w) \rangle$  to the  $MS$ . Similar to those illustrated in unsubscribe guard, the special token  $UAT^P(w)$  is computed as follows:  $UAT^P(w) = KH_{rk(MS)}(adId)$ , where  $adId$  denotes the advertisement identifier, namely, the  $r$ -component of the signature  $sig_{MS}^P(T(w))$ . Note that the use of  $UAT^P(w)$  ensures DoS attacks based on phony unadvertisements. Let  $\phi'(w)$  denote the original unadvertisement message for topic  $w$ .

$$\begin{aligned}\phi'(w) &= \langle \mathbf{publisher}, EQ, P \rangle, \langle \mathbf{topic}, EQ, w \rangle \\ uad(w) &= \langle sig_{MS}(T(w) \parallel P \parallel adId) \rangle \\ \phi(w) &= \langle \mathbf{publisher}, EQ, P \rangle, \langle \mathbf{topic}, EQ, T(w) \rangle, \langle sig, ANY, sig_{MS}(T(w) \parallel P \parallel adId) \rangle\end{aligned}$$

Upon receiving an unadvertise request from publisher  $P$ , the  $MS$  generates an unadvertisement permit  $uad(w)$  and send it back to the publisher  $P$ . The publisher  $P$  uses the

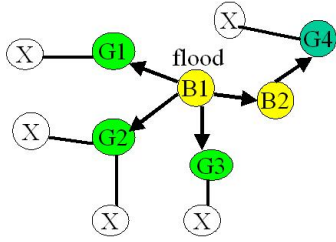
advertisement signature  $sig_{MS}^P(T(w) \parallel P \parallel adId)$  included in the permit to create a legal unadvertise request and submit it to the pub-sub overlay network. This signature (similar to unsubscription) is used by the routing nodes to check its authenticity and detect DoS attacks based on unadvertisement flooding.

### 3.4.7 Routing Guard

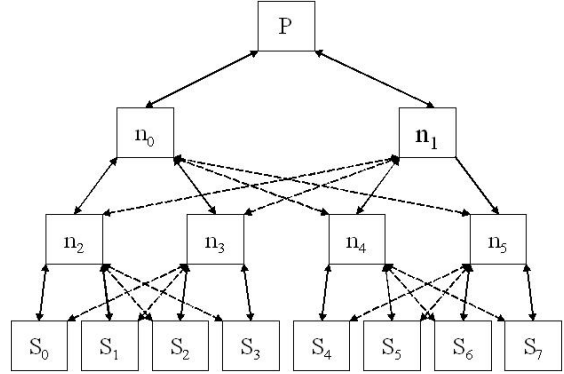
The pub-sub network nodes route messages based on tokens – the pseudonym for topics. Besides performing the functionality of a regular pub-sub node, we require the nodes to perform additional checks to ensure safety from DoS attacks. Now, we discuss the checks implemented by nodes to protect the pub-sub network from flooding-based DoS attacks.

EventGuard requires nodes on the pub-sub network to perform two security checks. The first check is based on signatures for maintaining sender authenticity and the second check is based on detecting duplicate messages. Subscriptions, unsubscriptions, advertisements and unadvertisements are verified for the  $MS$ 's signature. The publications are verified for its publisher's signature. Duplicates are checked using the  $r$ -component of the signature. Recall that we designate the  $r$ -component of the ElGamal signature as the message's identifier. When a node receives two subscriptions with the same identifier, it blocks the later one. With the guarantee of sender authenticity and the prevention of duplicate messages, no flooding attack could propagate beyond one good pub-sub node. Figure 20 illustrates this point. In Figure 20, a malicious (bad) node  $B1$  attempts a flooding based DoS attack to all its neighbor nodes. Observe that no invalid message (incorrect signatures) and no duplicate message from node  $B1$  would propagate beyond the non-malicious (good) nodes  $G1$ ,  $G2$ ,  $G3$  and  $G4$ . More importantly, none of the nodes marked  $X$  would be hit by this DoS attack. Thus, by deploying routing guards in the pub-sub network, EventGuard can effectively isolate the effect of flooding attacks.

We implement the routing guard (i.e., the two security checks on each routing node) in three steps. First, we require a node to maintain the history of identifiers previously seen by it. Second, we augment each EventGuard message with a timestamp that is signed by the  $MS$  (for advertisement, subscription, unadvertisement and unsubscription) or signed



**Figure 20:** Handling Flooding based DoS attacks in EventGuard



**Figure 21:** Constructing Resilient Networks: Thick lines represent links in the binary tree network and the dashed lines represent additional links added to binary tree network to make its  $ind = 2$

by the publisher (for a publication). Third, a non-malicious node blocks any message if the condition  $|ct - ts| > max\_delay$  is met, where  $ct$  is the current time,  $ts$  is the timestamp on a message, and  $max\_delay$  is a system defined parameter. Nodes only need to maintain a history of identifiers for a time duration of  $max\_delay$ . Note that  $max\_delay$  must account for time skew between nodes and routing and communication delays on the pub-sub network.

### 3.4.8 Rekeying

Authorization keys are capabilities issued to authorized subscribers. Hence, when a subscriber unsubscribes its subscriptions, an authorization is revoked, the authorization key needs to be changed and the new key has to be communicated to all authorized subscribers. We observe that changing the per topic authorization key each time when an authorized subscriber unsubscribes can be very expensive. As a result, EventGuard resorts to periodic rekeying.

We periodically change all per topic authorization keys by changing  $rk(MS)$ . Note that changing  $rk(MS)$  changes all the keys in the pub-sub system. More specifically, we perform a periodic rekeying operation as follows. We divide time into epochs of *epoch* time units (say, one month). All subscriptions and advertisements need to be renewed at the beginning of every time epoch. We number epochs with consecutive integers starting

from epoch number 0. The secret key used by the  $MS$  in the  $T^{th}$  epoch is derived from the primary secret key  $rk(MS)$  as  $rk(MS, T) = KH_{rk(MS)}(T)$ . The  $MS$  uses  $rk(MS, T)$  to replace  $rk(MS)$  during the  $T^{th}$  epoch for generating authorization keys. Hence, if a subscriber  $S$  unsubscribes for topic  $w$  in epoch  $T$ , it would still be able to read the contents of publications under topic  $w$  till the end of epoch  $T$  (but not after epoch  $T$ ).

Note that  $rk(MS, T)$  is used only for generating authorization keys. The  $MS$  always uses  $rk(MS)$  for signing subscriptions and advertisements. Also, all topic tokens, unsubscribe and unadvertise tokens are derived using the original secret key  $rk(MS)$ . Hence, tokens and special tokens do not change across epochs. Therefore, none of the subscription and advertisements disseminated into the pub-sub network needs to be changed every epoch. Our rekeying technique requires only the subscribers and publishers to obtain new keys from the  $MS$  every epoch. Thus periodic rekeying additionally facilitates the  $MS$  to bill the subscribers and the publishers for the next epoch.

### 3.5 *EventGuard: Scalable Key Management*

Most existing key management solutions for pub-sub networks use group key management protocols to manage subscriber grouped based on their subscriptions. However, given a flexible subscription filter based authorization model, every event can potentially go to a different subset of subscribers. In the worst case, for  $NS$  subscribers, there are  $2^{NS}$  subgroups, thereby making it infeasible to setup static groups for every possible subgroup. Although some optimizations have been proposed for dynamic groups such as key caching [66], the worst case key management cost remains at  $O(2^{NS})$  due to its inherent design.

In this chapter, we propose to improve past solutions to the key management problem using a completely different design philosophy. Our key management algorithms disassociate keys from subscriber groups and ensure that the key management cost is *independent* of the total number of the subscribers ( $NS$ ) in the pub-sub system. We achieve this in two steps: (i) First, we associate an authorization key  $K(f)$  with a subscription filter  $f$  and an encryption key  $K(e)$  with an event  $e$ . We use the encryption key  $K(e)$  to *encrypt* the secret attributes in an event  $e$  and the authorization key  $K(f)$  to *decrypt* the secret attributes

in a matching event  $e$ . (ii) We use hierarchical key derivation algorithms [118] to map the authorization keys and the encryption keys into a common key space. The mapping ensures that a subscriber can efficiently derive an encryption key  $K(e)$  for an event  $e$  using an authorization key  $K(f)$  for the subscription filter  $f$  if and only if the event  $e$  matches the subscription filter  $f$ . In this chapter, we present a detailed quantitative analysis of our approach and show that it incurs a small computation, communication and storage cost that is independent of the number of subscribers, thereby making our approach very efficient and scalable.

We have developed hierarchical key derivation algorithms for constructing key spaces for different types of publication-subscription matching, including: topic or keyword based matching, numeric attribute based matching, category or ontology based matching, and string prefix/suffix matching. For example, in a numeric attribute based key space construction algorithm, an authorization key  $K(f_1)$  associated with the filter  $f_1 = \langle \langle \text{topic}, EQ, \text{cancerTrail} \rangle, \langle \text{age}, >, 20 \rangle \rangle$  and an authorization key  $K(f'_1)$  associated with the filter  $f'_1 = \langle \langle \text{topic}, EQ, \text{cancerTrail} \rangle, \langle \text{age}, >, 30 \rangle \rangle$  must be capable of deriving the encryption key  $K(e_1)$  used for encrypting the message  $msg$  in event  $e_1 = \langle \langle \text{topic}, \text{cancerTrail} \rangle, \langle \text{age}, 35 \rangle, \langle \text{message}, msg \rangle \rangle$ . On the other hand, key  $K(f_1)$  should be capable of decrypting the message  $msg$  in event  $e'_1 = \langle \langle \text{topic}, \text{cancerTrail} \rangle, \langle \text{age}, 25 \rangle, \langle \text{message}, msg \rangle \rangle$ , but the key  $K(f'_1)$  should not. In this section, we illustrate our approach using numeric attribute based matching.

### 3.5.1 Numeric Attribute Based Matching

Numeric attribute based matching supports range conditions on numeric attributes. Given a numeric attribute, say `age`, we can construct a subscription filter  $f = \langle \text{age}, \in, (l, u) \rangle$  such that the filter  $f$  matches any event  $e = \langle \text{age}, v \rangle$  if and only if  $l \leq v \leq u$ , that is, the attribute `age` takes a value  $v$  that belongs to the range  $(l, u)$  (both end points inclusive). To enable secure publication-subscription matching, we associate an authorization key  $K(f)$  with every subscription filter  $f$  and an encryption key  $K(e)$  with every event  $e$  that satisfy the following properties:

- Given  $K(f)$  it should be computationally easy to derive a key  $K(e)$ , if  $v \in (l, u)$ .
- Given  $K(f)$  it should be computationally hard to derive a key  $K(e)$ , if  $v \notin (l, u)$ .

We construct keys that satisfy the above mentioned properties as follows. We map the authorization keys and encryption keys to the common key space using a numeric attribute key tree (NAKT). Given a subscription filter  $\langle \mathbf{age}, \in, (l, u) \rangle$ , we use the NAKT to derive a small set of authorization keys that corresponds to the attribute  $num$  with the range  $(l, u)$ , denoted by  $K_{(l,u)}^{num}$ , where  $num$  denotes the name of the numeric attribute. The NAKT enables one to easily (computationally) derive a key  $K_{(l',u')}^{num}$  from  $K_{(l,u)}^{num}$  if and only if  $l \leq l' \leq u' \leq u$ . For any event  $e = \langle \mathbf{age}, v \rangle$ , we derive the encryption key  $K(e) = K_{(v,v)}^{num}$  and encrypt the event  $e$  with  $K(e)$ . By the construction of the numeric attribute key tree it follows that  $K(e)$  is efficiently derivable from  $K(f)$  if and only if  $l \leq v \leq u$ .

**Constructing Numeric Attribute Key Tree (NAKT).** Now we need to discuss how to build the NAKT tree for a given numerical attribute  $num$  using hierarchical key derivation algorithms. Without loss of generality, we assume that the actual range of the numeric attribute  $num$  is  $(0, |R(num)| - 1)$ , where  $|R(num)|$  denotes the size of range  $R(num)$ . Given a numeric value  $v \in (0, |R(num)| - 1)$ , we map it to a key tree identifier  $ktid(v)$  which is a  $m$ -bit binary representation of the number  $\lfloor \frac{v}{lc(num)} \rfloor$  and  $m = \log_2(\frac{|R(num)|}{lc(num)})$ . Note that  $lc(num)$  denotes the smallest size of a subscription on numeric attribute  $num$ . Later in the section, we use  $lc(num)$  to trade-off performance and expressiveness of the numeric attribute based matching algorithm. The key tree identifiers are arranged in the form of a binary tree with depth  $m$ . Figure 22 shows a numeric attribute key tree for  $R(num) = (0, 31)$  and  $lc(num) = 4$ . Each element in the tree labeled  $ktid$  has two attributes: a key  $K_{ktid}^{num}$  and a range of numeric values  $v$  such that  $ktid(v)$  share the prefix  $ktid$ . The key tree is designed such that given a parent key all its children keys can be easily derived; but the converse is computationally infeasible.

Let the symbol  $\emptyset$  (*null*) be used to label the root element of a NAKT. We derive the authorization key for the root element corresponding to the key tree as  $K_{\emptyset}^{num} =$

$KH_{K(w)}(num)$ , where  $KH$  is a keyed pseudo-random function (approximated by HMAC-MD5 or HMAC-SHA1 [52]),  $K(w) = KH_{rk(KDC)}(w)$  is the authorization key for the topic  $w$ , and  $rk(KDC)$  denotes the secret key of the KDC. An example topic would be  $w = \text{cancerTrail}$  and numeric attribute  $num = \text{age}$ . For example, the key for the root element for the  $\text{age}$  numeric attribute is derived as  $K_{\emptyset}^{\text{age}} = KH_{K(\text{cancerTrail})}(\text{age})$ , where  $K(\text{cancerTrail}) = KH_{rk(KDC)}(\text{cancerTrail})$ . Then, we derive the key for an internal element with  $ktid = \xi \parallel b$  recursively as  $K_{\xi \parallel b}^{num} = H(K_{\xi}^{num} \parallel b)$ , for some  $\xi \in (0 + 1)^*$ ,  $b = 0$  or  $1$  and  $H$  is a one-way hash function (approximated by MD5 [80] or SHA1 [32]). Note that  $\parallel$  denotes string concatenation. For example,  $K_0^{\text{age}}$  is derived as  $K_0^{\text{age}} = H(K_{\emptyset}^{\text{age}} \parallel 0)$  and  $K_1^{\text{age}}$  is derived as  $K_1^{\text{age}} = H(K_{\emptyset}^{\text{age}} \parallel 1)$ . Having described how to construct the numeric attribute key tree, we now describe techniques to select an encryption key  $K(e)$  for an event  $e$  and an authorization key  $K(f)$  for a filter  $f$ .

**Encryption Key.** A publisher  $P$  constructs the encryption key for a numeric attribute in a given event  $e$  as follows:

$$\begin{aligned}
 e &= \langle \langle \text{publisher}, P \rangle, \langle \text{topic}, w \rangle, \langle \text{num}, v \rangle, \langle \text{message}, msg \rangle \rangle \\
 K(e) &= K_{ktid(v)}^{num}
 \end{aligned}$$

For example, a publisher  $P$  encrypts an event  $e = \langle \langle \text{publisher}, P \rangle, \langle \text{topic}, \text{cancerTrail} \rangle, \langle \text{age}, 22 \rangle, \langle \text{message}, msg \rangle \rangle$  is encrypted as follows. First,  $P$  identifies that the leaf node in the NAKT, which contains  $v = 22$  has an identifier  $ktid(22) = 101$  (see Figure 22). Then the publisher  $P$  encrypts the event  $e$  with the encryption key  $K(e) = K_{101}^{\text{age}}$ .

**Authorization Key.** A subscriber can subscribe for any range over the numeric attributes (limited by the least count  $lc(num)$ ). The subscription range may span one or more elements in the NAKT. Given a range  $(l, u)$  we identify the smallest set of elements in the NAKT  $SS$  that spans the range  $(l, u)$  using a simple depth first search starting from the root of the NAKT. Then, we divide the subscription for the range  $(l, u)$  into multiple subscriptions, one for each sub-range in the set  $SS$ . For example, the smallest set of elements in the NAKT that spans the range  $(8, 19)$  is  $SS = \{(8, 15), (16, 19)\}$ ; hence, we split a subscription on the range  $(8, 19)$  into two subscription ranges  $(8, 15)$  and  $(16, 19)$ . Without loss of generality,

we discuss how to generate authorization keys for subscriptions whose range spans exactly one element in the NAKT. The subscription  $f$  and its the authorization key  $K(f)$  are as shown below.

$$f = \langle \langle \mathbf{topic}, EQ, w \rangle, \langle \mathbf{num}, \geq, l \rangle, \langle \mathbf{num}, \leq, u \rangle \rangle$$

$$K(f) = K_{ktid(l,u)}^{num}$$

For example, a subscriber  $S$  subscribes for a filter  $f = \langle \langle \mathbf{topic}, EQ, \mathbf{cancerTrail} \rangle, \langle \mathbf{age}, \geq, 16 \rangle, \langle \mathbf{age}, \leq, 31 \rangle \rangle$  as follows. First, the authorization service identifies that element in the NAKT that matches the subscription range  $(16, 31)$  as  $ktid(16, 31) = 1$ . Then the authorization service sends  $S$  the authorization key  $K(f) = K_1^{\mathbf{age}}$ .

**Matching Publications with Subscriptions using the NAKT.** Given a publication with key tree identifier equal to  $ktid_\alpha$  a subscriber who has subscribed for key tree identifier equal to  $ktid_\phi$  does the following. The subscriber checks if  $ktid_\phi$  is a prefix of  $ktid_\alpha$ . If so, the subscriber derives the encryption key  $K_{ktid_\alpha}^{num}$  from the authorization key  $K_{ktid_\phi}^{num}$ . Note that the generation of children keys from its parent's key is computationally efficient because it uses a fast one-way hash function. However, it is computationally infeasible for a subscriber to derive the keys corresponding to its ancestors or its siblings. For example, given a publication with  $ktid_\alpha = 101$ , a subscriber who has subscribed for  $ktid_\phi = 1$  decrypts the message  $msg$  in a publication as follows. Given  $ktid_\alpha = 101$  and  $ktid_\phi = 1$ , the subscriber first extracts the suffix 01. Then,  $S$  derives  $K_{10}^{\mathbf{age}} = H(K_1^{\mathbf{age}} \parallel 0)$  and  $K_{101}^{\mathbf{age}} = H(K_{10}^{\mathbf{age}} \parallel 1)$ . Now,  $S$  can use  $K_{101}^{\mathbf{age}}$  to decrypt the secret attributes in the event.

**Number of Authorization Keys.** In general, if one uses a  $a$ -ary numeric attribute key tree ( $a \geq 2$ ), any subscription range can always be subdivided into no more than  $2(a-1) \log_a \left( \frac{|R(num)|}{lc(num)} \right) - 2$  sub-ranges. One can show that this is a monotonically increasing function in  $a$  (for  $a \geq 2$ ) and thus has a minimum value when  $a = 2$ . Thus, a binary tree is optimal and it requires no more than  $2 \log_2 \frac{|R(num)|}{lc(num)} - 2$  authorization keys for any given subscription range. In addition, one can also show that the average number of sub-ranges for a uniformly and randomly chosen subscription range of length  $\phi_R$  is  $\log_2 \frac{\phi_R}{lc(num)}$ . Observe that the number of keys is at most logarithmic in  $|R(num)|$ ; additionally one can

control the number of keys by tuning the parameter  $lc(num)$ .

The key distribution center (KDC) expends computing power to generate keys for subscribers during the authorization phase. One can show that the maximum and the average cost of generating authorization keys for a subscription is  $(4 \log_2 \frac{|R(num)|}{lc(num)} - 2)$  and  $(\log_2 \frac{|R(num)|}{lc(num)} + \log_2 \frac{\phi_R}{lc(num)} - 1)$  hash operations respectively. The publishers and the subscribers expend computing power to derive keys for encrypting/decrypting events. One can show that the maximum and the average cost of deriving the encryption/decryption keys is  $(\log_2 \frac{|R(num)|}{lc(num)})$  and  $(\log_2 \frac{\phi_R}{lc(num)})$  hash operations respectively. Note that these average assume that the subscription range is chosen uniformly and randomly over  $R(num)$ .

Tables 9 and 10 shows the maximum and the average number of keys, key generation cost and key derivation cost for different values of  $|R(num)|$  assuming  $lc(num) = 1$ . Observe that the number of authorization keys is very small. The key generation cost at the KDC is only of the order of few tens of microseconds and thus allowing the KDC to handle large subscription traffic. The key derivation cost is only a few microseconds, thereby adding minimal overhead to the throughput and latency of the published events.

We have described the design of our key derivation algorithm using numeric attribute based matching. We refer the readers to our technical report [5] for other types of publication-subscription matching and the techniques we have developed to handle complex subscriptions that include one or more of the above matching constraints combined using  $\wedge$  and  $\vee$  Boolean operators.

**Unsubscription by Rekeying.** In EventGuard, an authorization key  $K(f)$  act like a capability issued to authorize subscribers to read all events  $e$  that match the filter  $f$ . As described in our subscription model (see Section 3.2.1), all subscriptions are accompanied by a payment and are valid for one time epoch. We use a rekeying algorithm that is similar to the lazy revocation (epoch based periodic rekeying) algorithms used in several group key management protocols [121]. At the beginning of a new epoch, if the subscribers need to refresh their subscriptions then they must obtain new authorization keys from the KDC. To avoid flash crowds attempting to subscribe at the beginning of a new epoch, we evenly space out the epoch intervals on a per-topic basis. We also adaptively vary the length of the

$R$	# Keys	Key Gen ( $\mu s$ )	Key Derive ( $\mu s$ )
$10^2$	12	23.66	6.37
$10^3$	18	34.58	9.10
$10^4$	26	49.14	12.74

**Table 9:** Max Cost

$\phi_R$	# Keys	Key Gen ( $\mu s$ )	Key Derive ( $\mu s$ )
10	3.32	14.20	3.02
$10^2$	6.64	17.22	6.04
$10^3$	9.97	20.25	9.07

**Table 10:** Avg Cost:  $R = 10^4$

epoch on a per-topic basis using the subscription history. Detailed discussion on choosing the per-topic epoch length is outside the scope of this chapter.

**Multiple Publishers.** When multiple publishers publish on a common topic, it might be essential to ensure that the publications from a publisher  $P$  is not readable by another publisher  $P'$ . EventGuard handles this problem using a small modification to the authorization key  $K(w)$  for topic  $w$ . Instead of having a topic key shared across all users the KDC can generate per publisher authorization key for topic  $w$  as  $K^P(w) = KH_{rk(KDC)}(P \parallel w)$ . The KDC distributes  $K^P(w)$  to a publisher. The KDC uses  $K^P(w)$  to derive authorization keys for subscribers that subscribe to a topic  $w$  from publisher  $P$ . This only incurs almost no additional key generation cost. On the other hand, the subscriber group approach has to maintain separate groups for every publisher  $P$ .

### 3.5.2 Comparison with Subscriber Group Approach

#### 3.5.2.1 Overview

In this section, we first illustrate (using examples) there important benefits of our approach over the traditional subscriber group based approach in terms of the key management cost. We then use a formal quantitative analysis to derive theoretical lower bounds on the performance and scalability benefits offered by our approach.

**Number of Keys.** First, let us suppose that a subscriber  $S$  has subscribed for a range  $(0, R-1)$ . Using the subscriber group based approach the number of keys is bounded by the number of possible subscription ranges and the number of subscription groups:  $\min(\frac{R(R-1)}{2}, 2^{NS})$ . In contrast, EventGuard uses efficient key derivation algorithms to ensure the number of authorization keys is *independent* of the number of subscribers. Using the EventGuard approach the key server maintains only one key. The number of keys maintained by a subscriber  $S$  is at most logarithmic in  $R$ .

**Communication Cost.** Second, the subscriber group based approach would require changes to the groups and group keys whenever a new subscriber joins the system. For example, let subscriber  $S_1$  subscribe for a range  $(20, 30)$ . We have one group  $G = \{S_1\}$ . Let us suppose that a new subscriber  $S_2$  subscribes for a range  $(25, 40)$ , then we have three groups:  $G_1 = \{S_1\}$  (for the range  $(20, 25)$ ),  $G_2 = \{S_1, S_2\}$  (for the range  $(25, 30)$ ), and  $G_3 = \{S_2\}$  (for the range  $(30, 40)$ ). Observe that the group key server has to not only maintain more keys (computing and storage cost), but also update subscriber  $S_1$  with new group keys (communication cost). On contrary, our approach requires no key updates and thus incurs much lower communication costs. We use a quantitative analysis to show that the EventGuard can offer 2-3 orders of magnitude reduction in communication costs.

**KDC Scalability.** Third, in the subscriber group approach, the key server has to maintain all subscriptions made by all active subscribers in order to determine the key updates (as illustrated above). The EventGuard approach allows the key server to be *stateless* and ensures that the cost of handling a subscription request is very small (independent of  $NS$ ). In the EventGuard approach, the key server does not have to maintain information about active subscriptions and active subscribers or update any authorization key as more subscribers join the pub-sub network. The stateless nature of our key server allows us to distribute and *on-demand* replicate the key server it to handle bursty loads. Note that the key server replicas need no consistency and concurrency control since they share no common state other than the master key  $rk(KDC)$ ; recall that all authorization keys are derivable from  $rk(KDC)$ .

### 3.5.2.2 Quantitative Analysis

We now use an analytical model to compare the cost (messaging, computation and storage) of our approach versus the subscriber group approach (using lazy revocation). In order to make a fair comparison we assume that the time interval for lazy revocation in the subscriber group approach equals the length of one time epoch  $T$ . We assume an  $M/M/N$  model for subscribers [121] with  $\lambda$  denoting the arrival rate per inactive subscriber and  $\mu$  denoting the departure rate per active subscriber and  $N$  denotes the total number of subscribers

(active + inactive). One can show that the average number of active subscribers at any point in time is  $NS = N^* \frac{\lambda}{\lambda + \mu}$ . In steady state, the average rate of subscribers joining the system = the average rate of subscribers leaving the system =  $N^* \frac{\lambda \mu}{\lambda + \mu}$ . Let us assume that we have only one topic and one numeric attribute whose range is of size  $R$ . Without loss of generality we assume that the least count parameter is set to one. Let  $\phi_R$  denote the average size of a subscription range. We assume that the subscription ranges are chosen uniformly and randomly over  $(0, R - 1)$ .

In the subscriber group approach, when a new subscriber  $S$  joins the system the KDC has to determine the number of active subscribers  $NS_{overlap}$  who have an overlapping subscription range with  $S$ . A subscription for  $(x_s, x_s + \phi_R)$  by  $S$  overlaps with  $(x_{s'}, x_{s'} + \phi_R)$  by some other subscriber  $S'$  overlaps if  $|x_s - x_{s'}| \leq \phi_R$ , that is, if  $x_{s'}$  lies between  $x_s - \phi_R$  and  $x_s + \phi_R$ , the subscription ranges are guaranteed to overlap. Since subscription ranges are chosen uniformly, the probability that the subscription range of  $S$  overlaps with that of some active subscriber  $S'$  is  $\frac{2\phi_R}{R}$ . Since, the subscriptions of any two subscribers are independently chosen,  $NS_{overlap} \sim binomial(\frac{2\phi_R}{R}, NS)$  (if,  $\phi_R < \frac{R}{2}$ ). Note that if  $\phi_R \geq \frac{R}{2}$  the probability of overlap is one. Hence, the average number of overlapping subscribers  $NS_{overlap} = NS * \min(\frac{2\phi_R}{R}, 1)$ . In the following portions of this section, we assume that  $\phi_R < \frac{R}{2}$  to estimate  $NS_{overlap}$  (if  $\phi_R \geq \frac{R}{2}$ , then  $NS_{overlap} = NS$ ).

For every overlapping subscriber  $S'$ , the number of keys that need to be updated is: zero if  $S$  subscribes for a superset of  $S'$ , three to four if  $S$  subscribes to a subset of  $S'$  and two otherwise. The average number of keys that need to be updated per active subscriber with an overlapping range is two. This incurs a messaging cost of  $2 * NS_{overlap}$  keys. In addition, the new subscriber  $S$  has to be sent  $NS_{overlap}$  keys. Hence, the total messaging cost is  $3 * NS_{overlap} = 3 * NS * \frac{2\phi_R}{R}$ . Given a subscriber join rate of  $N^* \frac{\lambda \mu}{\lambda + \mu}$  and a time interval of length  $T$ , the total messaging cost is  $C_{subscribergroup} = N^* \frac{\lambda \mu}{\lambda + \mu} * T * 6 * NS * \frac{\phi_R}{R}$ .

In EventGuard the average number of authorization keys for a uniformly and randomly chosen subscription range  $\phi_R (> 1)$  is  $\log_2 \phi_R$ . Note that this cost is independent of the number of active subscribers  $NS$ . The total messaging cost for one time epoch is  $C_{EventGuard} = N^* \frac{\lambda \mu}{\lambda + \mu} * T * \log_2 \phi_R$ . Hence,  $C_{subscribergroup} : C_{EventGuard} = 6 * NS * \frac{\phi_R}{R * \log_2 \phi_R}$ . Observe

if  $\phi_R \ll R$ , that there is very little or almost no overlap between the subscription ranges from any two subscribers) then the subscriber group approach may perform better than the EventGuard approach. Observe that if  $\phi_R = R = 1$  presents the worst case scenario for the subscriber group approach since this would result in 100% overlap between any two subscription ranges.

One should observe that the uniform and random distribution for subscription ranges presents the best case scenario for the subscriber group approach since it increases the likelihood of smaller subscriber groups that contain mutually disjoint sets of subscribers. However, in most applications, subscriber interest follows auto-correlated heavy tailed distribution that allows a group of subscribers to share common interests. Formally, let us suppose that  $f(x)$  denotes a probability density function that a subscriber subscribes for a range  $(x, x + \phi_R)$ . Using the same analysis as above, the probability of overlap is given by  $op = \sum_x (f(x) * \sum_{y=x-\phi_R}^{x+\phi_R} f(y))$ . For the sake of simplicity let us suppose that  $\phi_R \ll R$  such that  $f(x)$  can be approximated to linear function over the small range  $(x - \phi_R, x + \phi_R)$ . In this case, the probability of overlap could be approximated to  $op = 2\phi_R * \sum_x f(x)^2$ . Given that  $\sum_x f(x) = 1$ , one can show that  $op$  is minimal when  $f(x) = \frac{1}{R}$  for all  $x$ , that is, if  $f(x)$  follows a uniform and random distribution. On the other hand, the EventGuard approach is *agnostic* to the distribution of subscriber interests. Hence,  $C_{subscribergroup} : C_{EventGuard} = 6 * NS * \frac{\phi_R}{R * \log_2 \phi_R}$  represents an *absolute minima* for the cost ratio.

Tables 11 and 12 summarizes an analytical comparison of our EventGuard approach against the subscriber group approach. Note that  $H$  denotes the computation cost for a one-way hash function and  $D$  denotes the cost of a decryption operation. Tables 13 and 14 shows the lower bound on cost ratio for varying subscription range  $\phi_R$  and the number of subscribers  $NS$  respectively. Table 13 shows that the subscriber group approach incurs at least 2-3 orders of magnitude higher cost than the EventGuard approach, clearly demonstrating the lack of scalability in the subscriber group approach. Table 14 indicates that for  $NS \leq 100$ , the group key management approach may perform better; although it does not scale well for higher values of  $NS$ . However, in our experimental section we use a more realistic heavy tailed distribution (to model groups of subscribers with common

	Join Message	Join Compute	Storage	Stateless
EventGuard	$\log_2 \phi_R$	$H * 2 \log_2 \phi_R$	1	Yes
Subscriber Group	$6 * NS * \frac{\phi_R}{R}$	-	$2 * NS$	No

**Table 11:** KDC Costs

	Join Message New Subscriber	Join Message Active Subscribers	Storage	Event Processing
EventGuard	$\log_2 \phi_R$	-	$\log_2 \phi_R$	$D + H * \log_2 \phi_R$
Subscriber Group	$2 * NS * \frac{\phi_R}{R}$	$4 * NS * \frac{\phi_R}{R}$	$2 * NS * \frac{\phi_R}{R}$	$D$

**Table 12:** Subscriber Costs

interests) and show that the group key management approach may offer marginally better performance only when  $NS \leq 8$ .

### 3.5.2.3 Performance Enhancement

In our implementation and experiments, we have used a *key caching* mechanism to further decrease the computational overhead in the EventGuard approach. When a subscriber  $S$  derives an encryption key  $K_{ktid_\alpha}^{num}$  from an authorization key  $K_\phi^{num}$  ( $ktid_\phi$  is a prefix of  $ktid_\alpha$ ) it caches all the intermediate keys computed in this process in its local key cache. Now, the subscriber  $S$  can compute an encryption key  $K_{ktid_{\alpha'}}^{num}$  from a cached key  $K_{ktid_{\phi'}}^{num}$  such that  $ktid_\phi$  is a prefix of  $ktid_{\phi'}$  which in turn is a prefix of  $ktid_{\alpha'}$ . Observe that computing  $K_{\alpha'}^{num}$  from  $K_{\phi'}^{num}$  costs  $H*(|ktid_{\alpha'}|-|ktid_{\phi'}|)$  and that from  $K_\phi^{num}$  costs  $H*(|ktid_{\alpha'}|-|ktid_\phi|)$  and  $|ktid_\phi| \leq |ktid_{\phi'}|$ , where  $|ktid|$  denotes the number of bits in  $ktid$ . Indeed  $K_{ktid_{\phi'}}^{num}$  would be the *optimal* cached key to derive  $K_{ktid_{\alpha'}}^{num}$  if  $ktid_{\phi'}$  is a prefix of  $ktid_{\alpha'}$  and  $|ktid_{\alpha'}|-|ktid_{\phi'}|$  is minimal. One should note that such optimizations are more meaningful when the events exhibit temporal locality. For example, let us consider stock quotes. Assuming that the stock price changes only nominally over small periods of time, two consecutive stock quote events are likely to carry prices that are numerically very close to one another.

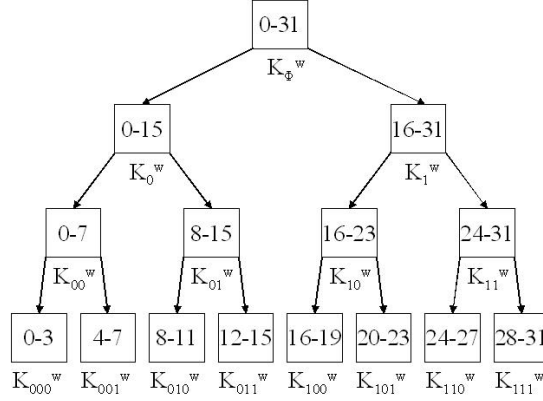
## 3.6 EventGuard: $r$ -Resilient Network Guard

The six security guards discussed so far can achieve message authenticity, confidentiality, integrity, and protect the pub-sub network from flooding-based DoS attacks. In addition, per topic token helps to alleviate selective message dropping attacks. However, they are

$\phi_R$	$C_{subscribergroup} : C_{EventGuard}$
10	1.81
$10^2$	9.04
$10^3$	60.18
$10^4$	451.81

$NS$	$C_{subscribergroup} : C_{EventGuard}$
10	0.09
$10^2$	0.90
$10^3$	9.04
$10^4$	90.36

**Table 13:** Theoretical Lower Bound:  $NS = 10^3$  and  $R = 10^4$       **Table 14:** Theoretical Lower Bound:  $\phi_R = 100$  and  $R = 10^4$



**Figure 22:** Key Tree: Range Queries on Numeric Attributes

incapable of handling random message dropping based attacks. In this section, we present techniques to restructure the pub-sub network in way that can effectively handle random message dropping based DoS attacks. We first define a  $r$ -resilient network.

**Definition** *r-resilient pub-sub network:* A pub-sub network is said to be  $r$ -resilient ( $0 < r < 1$ ) if  $r * 100\%$  of the messages are resilient to dropping attack.

There are two important design goals in constructing a  $r$ -resilient pub-sub network: (i) the pub-sub network must be resilient to message dropping attacks, and (ii) the communication cost should be minimal. We first discuss two network topologies that represent two extremities of the spectrum and then describe the EventGuard solution. The first network topology is a  $a$ -ary tree topology. The second network topology mirrors the propagation scheme used in Byzantine fault tolerant information dissemination [57]. The  $a$ -ary tree topology incurs minimum communication cost but is not strongly resilient to message dropping attacks. The BFT propagation algorithm incurs very high communication cost and is 100% resilient to message dropping attacks.

In this section, we proceed in three steps. First, we compare the communication cost between these two pub-sub network architectures. Second, we study the resilience of  $a$ -ary trees towards message dropping attacks. Third, we propose EventGuard network architectures that strike a trade-off between resilience to message dropping attacks and the communication cost.

### 3.6.1 Communication Cost

Let  $NS$  denote the number of subscribers in the system and  $N(w)$  denote the number of subscribers who have subscribed to topic  $w$ . In a  $a$ -ary tree network, we assume that each publisher corresponds to one  $a$ -ary tree and a publisher is the root of the tree, the subscribers who have matching subscriptions are the leaves of the tree and the pub-sub nodes are intermediate elements of the tree. The height  $h$  of the tree is given by  $\lceil \log_a NS \rceil$ . For simplicity we compute  $h$  by  $h = \log_a NS$  and assume that  $h$  is rounded up to an integer. Let  $M^{tree}(w)$  denote the communication cost (in terms of the number of messages) of propagating a publication on topic  $w$  from the publisher to the subscribers. Since the cost of sending the publication to any individual subscriber is lesser than or equal to  $h$ , the total cost  $M^{tree}(w) \leq hN(w)$ . Also, the publication message is never required to traverse any link of the tree more than once. Hence,  $M^{tree}(w) \leq \sum_{i=1}^h a^i = \frac{a}{a-1}(NS - 1)$ . Combining the two constraints, we have  $M^{tree}(w) \leq \min(hN(w), \frac{a}{a-1}(NS - 1))$ . Observe that the maximum communication cost for an  $a$ -ary tree occurs when  $N(w) = NS$  and  $M_{max}^{tree}(w) = \frac{a}{a-1}(NS - 1)$ .  $M_{max}^{tree}(w)$  is minimum when  $a = NS$ , that is, the publisher is directly connected to all the subscribers. In general, as the parameter  $a$  increases the expected communication cost decreases. However, the communication load on the publisher and pub-sub nodes increases with  $a$  since the publisher and the pub-sub nodes may have to forward an event to  $a$  children nodes.

The BFT propagation algorithm assumes that the number of malicious nodes ( $m$ ) is known. A non-malicious node accepts an event  $e$  as an authentic event if and only if it receives  $m+1$  identical copies of  $e$  from distinct  $m+1$  nodes. The key idea is that in any set of  $m+1$  nodes there is at least one non-malicious node, and thus if  $m+1$  distinct nodes report

an event  $e$  then  $e$  has to be authentic. In a BFT propagation scheme, irrespective of the network topology (grid, tree) used for propagation each subscriber has to minimally receive  $m + 1$  identical publication messages. Hence the communication cost, denoted by  $M^{bft}(w)$ , satisfies the following condition:  $M^{bft}(w) \geq (m + 1)N(w)$ . Assuming that  $NS = 1000$  and about 10% of the nodes are malicious,  $m = 100$ , we have  $M^{tree}(w) \leq \min(5N(w), 1332)$  (assuming a 4-ary tree:  $a = 4$  and  $h = \log_4 1000 \approx 5$  and  $\frac{a}{a-1}NS = 1332$ ) and  $M^{bft}(w) \geq 101N(w)$ . This implies that the communication cost in any BFT dissemination algorithm would be at least 20 times ( $\approx \frac{101N(w)}{5N(w)}$ ) the  $a$ -ary tree based algorithm. However, one should note that the BFT dissemination is completely resilient to message dropping attacks and is *unconditionally secure* (requires no digital signatures). In a wide-area network with node-to-node latency in the order of 70ms [125], it might be advisable to limit the communication cost while incurring addition signature verification cost (1-2ms per verification).

### 3.6.2 Resilience to Message Dropping Attacks

We have discussed the BFT propagation scheme and its complete resilience to message dropping attacks. In comparison, a simple  $a$ -ary tree-based network is vulnerable to a message dropping attack. A publication from the publisher successfully reaches a subscriber only if all the nodes on the routing path from the publisher to the subscriber are non-malicious. Assuming  $p$  denotes the fraction of nodes that are malicious and assuming that malicious nodes are randomly distributed on the network. The probability that a publication reaches a subscriber is  $Pr(succ) = (1 - p)^h$ . Even when  $p = 10\%$ , with  $h = 5$  we find that the probability of a successful delivery of a publication is only 0.59. This implies that 10% malicious nodes are able to harm about 41% of the subscribers. One way to increase  $Pr(succ)$  is to increase  $a$  (consequently decrease  $h$ ). However, as we have pointed out earlier, as  $a$  increases the load on the publisher and the nodes on the pub-sub network increases, thereby harming the scalability of the system.

The key problem with the tree-based topology is that there is only one *independent path* from a publisher to a subscriber [97]. Informally, two paths  $Q_1$  and  $Q_2$  are independent if they share no node other than their source and their destination node. If we have *ind*

independent paths between a publisher  $P$  and a subscriber  $S$ , then  $ind$  malicious nodes (one per independent path) could completely block any communication between  $P$  and  $S$ . The BFT propagation scheme uses  $m + 1$  independent paths to propagate the publication thereby ensuring that at least one independent path is devoid of malicious nodes. Note that using an arbitrary peer-to-peer topology for the pub-sub network does not directly entail the existence of multiple independent paths [97].

### 3.6.3 Low Cost Resilient Networks

In pub-sub systems one may not require absolute guarantee of message delivery at all time. This permits us to trade-off resilience with communication cost. We modify a  $a$ -ary tree such that it has  $ind$  independent paths while increasing the communication cost by not more than a factor of  $ind$  ( $ind \leq a$ ). For simplicity, we illustrate our technique by modifying a binary tree network to yield a network with  $ind = 2$ .

Figure 21 shows the key idea behind constructing a resilient event dissemination networks  $G^2$ . Note that  $d$  refers to the depth of a node, with root (publisher) at depth 0 and the leaves (subscribers) at depth  $h$ . For any node  $n$ , let  $parent(n)$  denote the parent of node  $n$  and  $sibling(n)$  denote an immediate left or right sibling of node  $n$ . EventGuard's resilient network adds one additional edge to every subscriber and every node in the system. Concretely, for every node  $n$  we add an additional edge from  $n$  to  $sibling(parent(n))$ . We now claim that the resilient network  $G^2$  has the following property.

**Claim 3.6.1** *The resilient network  $G^2$  has  $ind = 2$  independent paths from the publisher  $P$  to every subscriber in the system.*

We prove Claim 3.6.1 using Theorem 3.6.2 which explicitly constructs two independent paths from the publisher (root) to any subscriber (leaf) on the resilient network.

**Theorem 3.6.2** *Let  $Q = \langle P, n_1, n_2, \dots, n_d, S \rangle$  denote a path from the publisher  $P$  to some subscriber  $S$  in the original tree based network. Then,  $Q_1 = Q$  and  $Q_2 = \langle P, sibling(n_1), sibling(n_2), \dots, sibling(n_d), S \rangle$  are two independent paths from  $P$  to  $S$  in the resilient network.*

**Proof** First, we show that the path  $Q_2$  exists (path  $Q_1 = Q$  exists trivially). We show that for any  $1 \leq i \leq d$ , there exists an edge from  $sibling(n_i)$  to  $sibling(n_{i+1})$ . From path  $Q_1$  we know that  $n_i$  is the parent of node  $n_{i+1}$ . Hence,  $n_i$  is the parent of node  $sibling(n_{i+1})$ . By the construction of our resilient network, we add an edge from any node  $n$  to  $sibling(parent(n))$ . Hence,  $sibling(n_{i+1})$  is connected to  $sibling(n_i)$  (since,  $n_i = parent(sibling(n_{i+1}))$ ).

Second, we show that  $\{n_1, n_2, \dots, n_d\} \cap \{sibling(n_1), sibling(n_2), \dots, sibling(n_d)\} = \emptyset$ . First, for any  $1 \leq i \leq d$ ,  $n_i \neq sibling(n_i)$ . Second, for any two nodes  $n_i$  and  $n_j$   $1 \leq i, j \leq d$  such that  $i \neq j$ ,  $n_i \neq n_j$  since the node  $n_i$  is at depth  $i$  from the root, while  $n_j$  is at depth  $j$  from the root ( $i \neq j$ ). Hence, the paths  $Q_1$  and  $Q_2$  are independent. ■

One can easily extend this network construction scheme for any  $ind \leq a$ . Construct a resilient network  $G^{ind}$  by connecting any node  $n$  to  $parent(n)$  and  $ind - 1$  distinct siblings of  $parent(n)$  (these siblings indeed exist since  $ind \leq a$ ).

**Claim 3.6.3** *The resilient network  $G^{ind}$  has  $ind$  independent paths from the publisher  $P$  to every subscriber in the system.*

**Proof** The proof for Claim 3.6.3 follows the same lines as that for Claim 3.6.1. ■

**Claim 3.6.4** *The resilient network  $G^{ind}$  incurs  $ind$  times the communication cost of  $G^1$ .*

**Proof** The proof follows from the construction of independent paths in Theorem 3.6.2. ■

Due to space constraint we omit the full proof for Claims 3.6.3 and 3.6.4. As we increase  $ind$ , the communication cost increases by a factor  $ind$ . However, we believe that  $ind = 2$  would suffice for most practical pub-sub networks. Assuming  $p$  denotes the fraction of nodes that are malicious and assuming that the malicious nodes are randomly distributed on the network. The probability that publication reaches a subscriber is  $Pr(succ) = 1 - (1 - (1 - p)^h)^{ind}$ . With  $p = 0.1$  and  $ind = 2$  we would require  $h \leq 3.66$  for  $Pr(succ) \geq 0.9$ . For a pub-sub network with each publisher having 1000 subscribers as the upper bound, this would translate to  $a = 7$  (7-ary tree). On the other hand, achieving the same level of resilience with  $ind = 1$  would require  $h \leq 1$  and thus  $a = NS$ . Recall that as  $a$  increases,

	$MS$ (ms)	publisher (ms)	subscriber (ms)	node (ms)
subscribe	$1.44 + 0.00117 *  w $	-	1.7	1.7
unsubscribe	3.14	-	1.7	1.7
publish	-	$1.4 + ( pbl  + 16m) * 0.0001$	$1.7 + ( pbl  + 16) * 0.0001$	1.7
advertise	$1.4 + 0.00117 *  w $	1.7	-	1.7
unadvertise	3.14	1.7	-	1.7

**Table 15:** Computation Overheads for EventGuard Operations:  $w$  is some topic,  $pbl$  is a publication, and  $m$  denotes the number of topics marked on message  $pbl$

the load on the publisher and the nodes on the pub-sub routing path increases and affects the system’s scalability. In general, our technique can be employed to construct a  $r$ -resilient network with  $Pr(succ) = r$  by carefully choosing  $ind$  and  $a$ .

### 3.7 EventGuard Evaluation

We have implemented EventGuard on top of the Siena pub-sub core [16]. We used a Java based implementation of Siena [15] and added EventGuard extensions to it in the form of an *EventGuard package*. No changes were made as such to the Siena pub-sub core (e.g., the content-based routing and event matching algorithms).

We evaluate our EventGuard prototype implemented on the Siena pub-sub core in two steps. We first present some micro-benchmarks to quantify the overhead of EventGuard mechanisms and measure the performance and storage overheads at the  $MS$ , a publisher, a subscriber and a node. Then we present macro-benchmarks to quantify the overhead of the entire system. We measure changes in throughput, latency of the pub-sub network as an effect of EventGuard mechanisms. We also quantify the effect of EventGuard’s resilience to DoS attacks.

#### 3.7.1 Micro-Benchmarks

In this section, we analytically estimate the amount of computational and storage overhead due to EventGuard on the pub-sub system. All our measurements were made on a 900MHz Intel Pentium III running RedHat Linux 9.0 using Sun Java 1.5.0.

We perform an analytical estimate on the computational time for subscriptions, publications and unsubscriptions. The cost for an advertisement is very similar to that of subscriptions and the cost for unadvertisement is equivalent to that of an unsubscription.

subscription (Bytes)	unsubscription (Bytes)	publication (Bytes)	advertisement (Bytes)	unadvertisement (Bytes)
128	128	$128 + 16m$	128	128

**Table 16:** Message Size Overhead due to EventGuard including only those messages sent on the pub-sub network:  $m$  denotes the number of topics marked on the publication

$MS$ (Bytes)	publisher (Bytes)	subscriber (Bytes)	node (Bytes)
64	$180 \text{ per adv} + HT_{size}$	$180 \text{ per sub} + HT_{size}$	$HT_{size}$

**Table 17:** EventGuard Storage Overhead:  $HT_{size}$  denotes the total size of the hashtable maintained for detecting flooding based DoS attacks ( $HT_{size}$  is at most a few tens of KBs)

We analyzed the cost of these operations at all four entities: a publisher, a subscriber, the  $MS$  and a pub-sub node. We also analyzed the messaging and storage cost at these four entities. Tables 15, 16 and 17 summarizes the results obtained in this section. **Subscription.** The cost of a subscription at the  $MS$  includes the computation of key  $K(w)$ , token  $T(w)$ , special token  $UST(w)$  and an ElGamal signature on  $T(w)$  and the current timestamp  $ts$ . Since, the topic  $w$  is typically a short string, the cost of computing the key  $K(w)$  (using HMAC-MD5) is  $0.67 * |w|\mu s$ . The cost of computing token  $T(w)$  from  $K(w)$  (using MD5) is  $0.5 * |w|\mu s$ . The cost of computing special token  $UST(w)$  (using HMAC-MD5) is  $0.67 * |sig_r|\mu s = 42.9\mu s$ , where  $sig_r$  denotes the  $r$ -component of the  $MS$ 's ElGamal signature (note that  $|sig_r| = 512 \text{ bits} = 64 \text{ Bytes}$ ). The cost of computing an ElGamal signature is 1.4ms. Hence, the total cost per subscription topic (dominated by the signature computation time) is about  $1.44\text{ms} + 1.17 * |w|\mu s$ .

The cost of a subscription at the subscriber includes only the signature verification time. The cost of verifying an ElGamal signature is about 1.7ms. The cost of a subscription at a node in the pub-sub network is the cost required to process this subscription, which equals to the sum of the cost of verifying the  $MS$ 's signature and the cost of detecting duplicate identifiers to protect the pub-sub network from subscription-flooding based DoS attacks. Our experiments show that the cost of verifying duplicates is negligible ( $< 10\mu s$ ) when compared to the signature verification time (1.7ms). Further, our experiments show that the cost of processing a subscription at a node in EventGuard is only marginally higher than basic Siena ( $< 50\mu s$ ). Note that a publisher incurs no direct cost for a subscription.

**Unsubscription.** The cost of an unsubscription at the *MS* includes the verification of special token  $UST(w)$ , the verification of *MS*'s signature on the corresponding subscription token and the generation of an unsubscription permit. The cost of verifying the special token requires the computation of one keyed hash on the  $r$ -component of the *MS*'s signature. As shown in the case of subscription, this costs  $42.9\mu s$ . The cost of verifying an *MS*'s signature adds  $1.7ms$  and the cost of generating an unsubscription permit adds  $1.4ms$ . Hence, the total cost of an unsubscription at the *MS* is  $3.14ms$ .

The cost of an unsubscription at a subscriber includes only the signature verification time, which costs  $1.7ms$ . The cost of a unsubscription at a node in the pub-sub network is the cost required to process an unsubscription, which can be computed by the sum of the cost of verifying the *MS*'s signature and the cost of detecting duplicate identifiers to protect the network from DoS attacks based on unsubscription-flooding. Our experiments show that the cost of verifying duplicates is negligible when compared to the signature verification time.

**Publication.** The cost of a publication at its publisher includes the cost of encrypting the publication  $pbl$  with some random key  $K_r$  and the cost of encrypting  $K_r$  with  $K(w_i)$  for every topic  $w_i$  ( $1 \leq i \leq m$ ) marked on the publication. The total encryption time is  $(|pbl| + m * |K_r|) * 0.1\mu s = (|pbl| + 16m) * 0.1\mu s$  (note that  $|K_r| = 16$  Bytes). Computing the publisher's signature adds an additional  $1.4ms$ .

The cost of a publication at a subscriber includes the cost of checking the publisher's signature, the cost of decrypting the random key  $K_r$  and the cost of decrypting the message using key  $K_r$ . The total decryption time is  $(|pbl| + |K_r|) * 0.1\mu s = (|pbl| + 16) * 0.1\mu s$  (note that  $|K_r| = 16$  Bytes). Verifying the publisher's signature adds an additional  $1.7ms$ .

The cost of a publication at a node includes only the signature verification time. Similar to subscription, our experiments show that the cost of processing a publication at a node in EventGuard is only slightly higher than the cost of using basic Siena. Note that the *MS* is not involved directly in the publication process. This largely reduces the aggregate load on the *MS* as publications are considered by many applications as the most common operation on a pub-sub network.

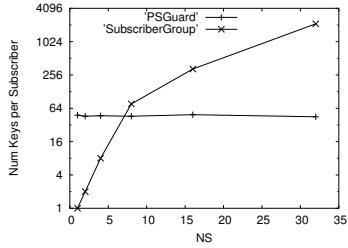
**Messaging Overhead.** We now study the overhead added in terms of the length of a message due to EventGuard. For subscriptions and advertisements, the primary overhead is due to the  $MS$ 's signature which is about 128 Bytes.

For publications, EventGuard adds the following overheads. First, the publisher's signature costs 128 Bytes. Second, the encrypted random key costs 16 Bytes. Third, the random key is encrypted by the topic's encryption key. This adds 16 Bytes for every topic included in the publication. The aggregate publication overhead may turn out to be quite significant if the published message is itself very small. On the other hand even if the published message is of the order of a few KBytes, the relative overhead added due to EventGuard turns out to be extremely small.

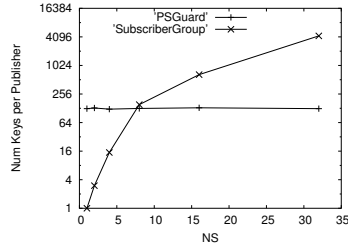
**Storage Overhead.** EventGuard requires publishers, subscribers and the  $MS$  to store additional information such as keys and tokens. The  $MS$  has the least storage overhead as it is required to store only the secret key  $rk(MS)$  (about 64 Bytes).

A subscriber has to store the key  $K(w)$  (16 Bytes), the token  $T(w)$  (16 Bytes), the special token  $UST(w)$  (16 Bytes), subscription time stamp  $ts$  (4 Bytes) and the  $MS$ 's signature (128 Bytes) for each subscription token  $w$ . Thus, the total per topic storage overhead is 180 Bytes. Further, the subscriber maintains a hashtable to store the publication identifiers (the  $r$ -component of the publisher's signature) in the near past ( $max\_delay$ ) to detect flooding based DoS attacks. The size of this hashtable obviously depends on the number of publications received by the subscriber in the last  $max\_delay$  time units. Our experiments show that this hashtable is typically small, with its size ranging from 100 Bytes to a few KBs.

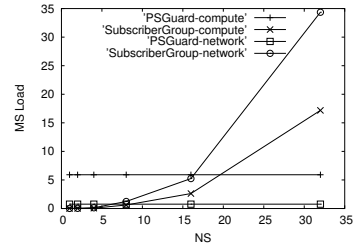
The storage overhead at a publisher is very similar to the storage overhead at a subscriber. However, the subscription identifier based hashtable maintained at the publisher is typically much smaller ( $< 1$  KB) than the publication identifier based hashtable at the subscribers, since we have number of subscriptions  $\ll$  number of publications. A node in the pub-sub network maintains two hash tables, one for subscriptions and one for publications for detecting flooding based DoS attacks. Our experiments show that the size of the subscription identifier based hashtable is usually very small ( $< 1$  KB) and the size of the



**Figure 23:** Num Keys per Subscriber



**Figure 24:** Num Keys per Publisher



**Figure 25:** KDC Load

publication identifier based hashtable is at most a few tens of KBs.

In summary, the performance overhead added by EventGuard is mostly dominated by digital signatures (2ms). However, in a wide-area network where the network latencies are in the order of 70ms [125] the percentile overhead added by EventGuard is significantly smaller.

### 3.7.2 Key Management

This section compares our key management algorithms with the subscriber group based approach in terms of the number of keys, communication and computation cost. We simulated 128 topics, with the popularity of each topic varying according to a Zipf-like distribution [74]. Each subscriber subscribed for 32 topics chosen from the set of 128 topics using the Zipf distribution. Amongst 128 topics, 32 were numeric attributes, 32 were category attributes, 32 were string attributes and the rest 32 were simple topics (see Section 3.5.1 for examples). Numeric attributes had a range of size 256 units and a least count of 4 units; the subscription range was chosen using a Gaussian distribution with mean 128 and a standard deviation 32. Hence, the number of elements in the numeric attribute tree was 127 and the height of the numeric attribute tree was 6. Category trees were for height 4 and the number of children for each non-leaf element was chosen uniformly and randomly between 2 to 4. The average number of elements in a category tree was 82. The length of the string attributes were Zipf distributed between 1 and 8. Each publication message was assumed to be 256 Bytes long.

**Number of Keys.** Figure 23 shows the average number of keys maintained per subscriber

as the number of subscribers  $NS$  varies. Recall that the `SubscriberGroup` approach uses group key management techniques on subscriber groups [66] that require  $2^{NS}$  keys in the worst case. `PSGuard` requires a small and constant number of keys per subscriber that is independent of  $NS$ . Even for 32 subscribers, the number of keys per subscriber using the `SubscriberGroup` approach is about 40 times larger than the `PSGuard` approach. `PSGuard` achieves significant reduction in the number of keys, while incurring a computational overhead for running the key derivation algorithms on the publisher and the subscribers. In our later experiments we show that the cost of key derivation is very small compared to wide-area network latencies thereby making it easily affordable. Figure 24 shows the average number of keys maintained per publisher as  $NS$  the number of subscribers varies. The trends shown in Figure 24 are very similar to that in 23.

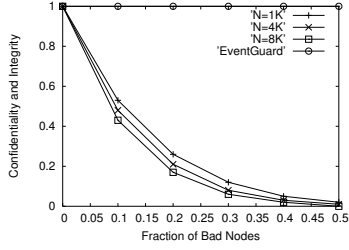
**KDC Load.** Figure 25 shows the computing and network cost on the key server using `SubscriberGroup` based approach and `PSGuard`. Computing cost (measured in milliseconds) shows the average cost of group key management in `SubscriberGroup` and the cost of key derivation algorithm in `PSGuard` when a new subscriber joins the system. The cost incurred by the `SubscriberGroup` increases dramatically with  $NS$ , while that incurred by the `PSGuard` approach is a small constant that is independent of  $NS$ . Networking cost (measured in KBytes) shows the average cost of communicating the updated group key in `SubscriberGroup` and the cost of delivering the authorization keys in `PSGuard`. Similar to computing cost, `PSGuard` incurs a small and constant networking cost, while that of `SubscriberGroup` explodes with  $NS$ .

### 3.7.3 Macro-Benchmarks

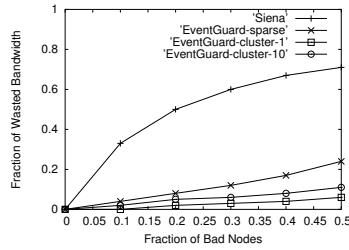
In this section, we present two sets of macro-benchmarks for `EventGuard`. The first set of experiments is simulation based. The second set of experiments is obtained from our prototype implementation of `EventGuard` on Siena pub-sub core.

#### 3.7.3.1 Simulation based Experiments

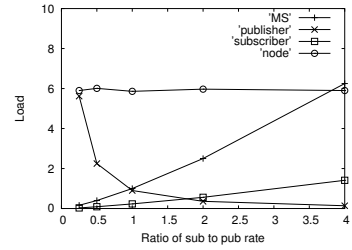
In this section, we present performance numbers from simulation based experiments on `EventGuard`. First, we present the improvements on message confidentiality and integrity



**Figure 26:** Confidentiality and Integrity



**Figure 27:** Flooding-based DoS Attack



**Figure 28:** *MS* Load

through EventGuard. Second, we measure the throughput of the system in the presence of malicious nodes. Third, we show the average load on the *MS*, the publisher, the subscriber and the nodes as we vary the subscription and publication rate. Fourth, we demonstrate the resilience of the pub-sub network architecture used in EventGuard against message dropping-based DoS attacks.

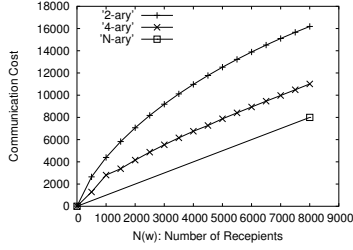
**Simulation Setup.** We used GT-ITM [125] topology generator to generate an Internet topology consisting of 4K nodes. We linked these nodes using open TCP connections to form a binary tree based hierarchical topology. The latencies for links were obtained from the underlying Internet topology generated by GT-ITM. The round trip times on these links varied from 24ms to 184ms with mean 74ms and standard deviation 50ms. We simulated 32 publishers and  $NS=8K$  subscribers. The publishers and subscribers were randomly connected to one leaf node in the pub-sub network. We used discrete event simulation [36] to simulate the function of the pub-sub network. All experimental results presented in this section were averaged over 5 independent simulation runs. We simulated 128 topics, with the popularity of each topic varying according to a Zipf-like distribution [74]. Each publisher publishes on 16 topics (randomly picked from the set of 128 topics) and each subscriber subscribes for 4 topics.

**Confidentiality and Integrity.** Figure 26 shows the fraction of messages that violate their confidentiality and integrity when in transit between a publisher and its subscribers with different fractions of malicious nodes ( $p$ ) and different values of  $NS$  (number of subscribers). We assume that a message loses its confidentiality and integrity as soon as it transits one bad node in the pub-sub network. Observe that when  $p$  is small, even a small

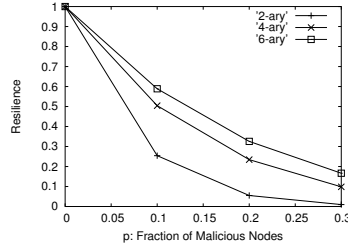
increase in  $p$  results in a heavy loss of message confidentiality and integrity. Note that as  $NS$  increases, the height of the binary tree network increases and so does the probability that at least one bad node appears on a path from a publisher to its subscribers. On the contrary, EventGuard is capable of preserving the confidentiality and integrity of all messages irrespective of the number of malicious nodes in the system.

**Flooding-based DoS Attack.** Figure 27 shows the fraction of network bandwidth expended on flooded messages as the fraction of malicious nodes ( $p$ ) varies with  $NS=8K$  subscribers. We assume that every malicious node performs a publication flooding-based DoS attack at the rate of 100 messages per unit time. We assume that each publisher publishes at the rate of 25 publications per unit time. We consider two cases: Case one wherein the malicious nodes are uniformly distributed throughout the pub-sub network (EventGuard-sparse in Figure 27); and Case two wherein malicious nodes form  $k$  clusters in the pub-sub network (EventGuard-cluster- $k$  in Figure 27). When malicious nodes are clustered together on the pub-sub network, we found that the loss in throughput for EventGuard is relatively much smaller. This is because EventGuard ensures that no flooding attack propagates beyond one non-malicious pub-sub node. Hence, if the malicious nodes are bunched together, they cannot significantly affect other non-malicious nodes in the system. Recall Figure 20, no flooding by either of the two malicious nodes B1 or B2 propagates beyond non-malicious nodes G1, G2, G3 and G4. Observe that if B2 were attached to some other part of the pub-sub network, then it could perform a flooding based DoS attack on a different set of non-malicious neighbor nodes that does not overlap with that of B1.

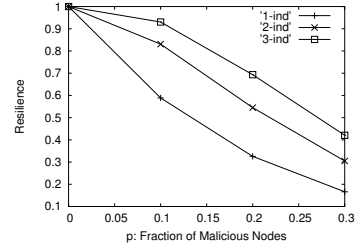
**Load.** Figure 28 shows the relative computational load on the  $MS$ , the publisher, the subscriber and a pub-sub node as we vary the rate of subscriptions, unsubscriptions and publications keeping the aggregate rate a constant (we do not consider advertisement and unadvertisement costs in this experiment). The computation load for basic operations were obtained from Table 15. We set the subscription rate to be equal to unsubscription rate so as to ensure that the average number of active subscriptions in the system is almost a constant. Note that only the control operations on subscriptions and unsubscriptions involves the  $MS$ . Hence, if a pub-sub network is largely dominated by publications (which



**Figure 29:** Communication Cost Vs Number of Receipients  $N(w)$



**Figure 30:** Resilience Vs  $a$  with  $ind = 1$



**Figure 31:** Resilience Vs  $ind$  with  $a = 6$

is true in most cases) then the relative load on the  $MS$  would be very small. If the load on a  $MS$  is not acceptable, EventGuard mechanisms easily permits one to add additional meta-servers. The fact that the meta-servers do not have to interact with one another makes it possible for one to build an efficient *load balancing* system to handle the  $MS$  load and vary the number of active meta-servers *on-demand*.

Observe that the load on a node remains almost a constant as it depends only the aggregate rate of subscriptions, unsubscriptions and publications. On the other hand the relative load on a publisher decreases as the publication rate decreases; this is because a publisher is not involved in subscribe and unsubscribe operations. Subscriber load is typically much smaller than the average node load because the number of publications delivered to a subscriber is very small when compared to the total number of publications sent on the pub-sub network. Recall that only those publications that match a subscriber’s subscriptions are delivered to the subscriber.

**Selective and Random Dropping Attack.** We now report the experimental results on the effectiveness of using the  $r$ -resilient pub-sub networks against message dropping attacks. Our first experiment measures communication cost versus  $a$  (for an  $a$ -ary tree network). The second and third experiments measure the network resilience as a function of  $p$  (the fraction of malicious nodes in the network).

Figure 29 shows communication cost for publishing an event under topic  $w$  versus  $N(w)$  for different values of  $a$  with  $ind = 1$ , where  $N(w)$  denotes the number of subscribers for topic  $w$ . Note that a resilient network constructed by modifying an  $a$ -ary tree increases

the communication cost by a factor  $ind$  (for some  $1 < ind \leq a$ ). Hence, the communication cost for an  $a$ -ary  $ind$  independent path network can be obtained by simply multiplying the corresponding cost for an  $a$ -ary tree network by  $ind$ . Observe that the communication cost decreases as  $a$  increases. Also note that  $a = NS$  minimizes the communication cost but imposes heavy load on the publisher and the pub-sub nodes (load is proportional to  $a$ ).

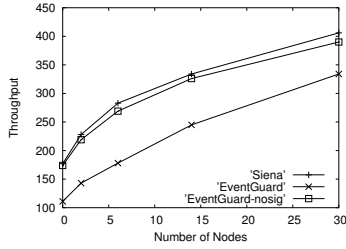
Figure 30 and 31 shows the resilience of the pub-sub network versus  $p$  (fraction of malicious nodes) for different values of  $a$  and  $ind$  respectively. Resilience is measured in terms of the ratio of the number of subscribers that receive an event on topic  $w$  to  $N(w)$ , averaged over all topics. Observe from Figure 30 that one can improve resilience by increasing  $a$  at the cost of publisher load. This is equivalent to decreasing the network's height  $h$  thereby, making the network shallow and broad. Figure 31 shows that one can improve resilience by increasing  $ind$  at the cost of the overall communication cost. A careful selection of parameters  $ind$  and  $a$  is required to strike a balance between resilience, communication cost and publisher load.

### 3.7.3.2 Implementation based Experiments

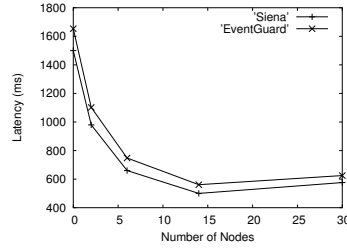
In this section, we present performance measurements from our prototype implementation of EventGuard on Siena pub-sub core. First, we present measurements on the loss in throughput and the increase in latency in publications due to EventGuard. Second, we measure the effectiveness of EventGuard against flooding based DoS attacks.

**Experimental Setup.** Our implementation of EventGuard is built on top of Siena pub-sub core. We ran this implementation of EventGuard on eight machines each with 8-processor (550 MHz Intel Pentium III Xeon processors running RedHat Linux 9.0) connected via a high speed LAN. We simulate the wide-area network delays obtained from the GT-ITM topology generator. We ignored the LAN delays as they measured only a few tenths of a millisecond.

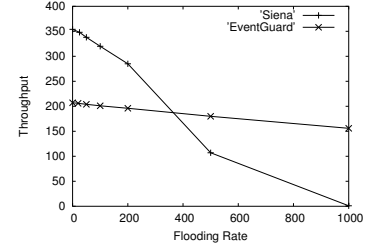
We used GT-ITM [125] topology generator to generate an Internet topology consisting of 63 nodes. The latencies for links were obtained from the underlying Internet topology generated by GT-ITM. The round trip times on these links varied from 24ms to 184ms with



**Figure 32:** Throughput



**Figure 33:** Latency



**Figure 34:** Resilience to Flooding-based DoS Attacks

mean 74ms and standard deviation 50ms. The tree’s root node acts as the publisher and its leaf nodes act as subscribers for this pub-sub network (32 subscribers and one publisher). We constructed complete binary tree topology using different number of nodes (0, 2, 6, 14, 30) and linked these nodes using open TCP connections to form the pub-sub network. The subscribers were uniformly distributed among all the leaf nodes.

**Throughput.** We measured the throughput in terms of the maximum number of publications per second that can be handled by the pub-sub system with and without EventGuard (EventGuard-nosig). We measured the maximum throughput as follows. We engineered the publisher to generate publications at the rate of  $q$  publications per unit time. In each run of this experiment, the rate  $q$  was fixed. We monitored the number of outstanding publications required to be processed at every node. If at any node the number of outstanding publications monotonically increased for five consecutive observations, then we conclude that the node is saturated and the experiment aborted. We iteratively vary  $q$  across different experimental runs to identify the minimum value of  $q_{min} = throughput$  such that some node in the pub-sub network is saturated.

Figure 32 shows the maximum throughput versus the number of nodes in the pub-sub network for EventGuard and basic Siena for simple subscriptions. The increase in throughput with the number of nodes shows the scalability of EventGuard. Note that as the number of nodes increases, the number of subscribers connected to one leaf node decreases, thereby increasing the effective throughput. However, as the number of nodes becomes increasingly larger than the number of subscribers the throughput does not increase any further, since

this simply results in underutilized nodes. The main overhead in EventGuard arises due to the verification of ElGamal signatures (1.7ms). We also measured the overhead in the absence of this signature verification at every node in the pub-sub network (EventGuard-nosig in Figure reftab-thruput). We found that the overhead was lesser than 5%. We are currently exploring faster signature algorithms to replace ElGamal.

**Latency.** We measured latency in terms of the amount of time it takes from the time instant a publication is published till the time it is available to the subscriber (in plain-text). The latency was measured keeping the throughput at its highest (see Figure 32). Figure 33 shows latency Vs number of nodes for EventGuard and basic Siena.

Observe that the latency first decreases and then increases. Initially, as the number of nodes increases, the number of subscribers assigned to each leaf node decreases. This consequently decreases the load on a node and thus decreases the latency. However, as the number of nodes increases, so does the height of the dissemination tree. An increase in height by one incurs an additional latency of 70ms (network latency), thereby increasing the overall latency. While the throughput always increases (until it saturates) with the number of nodes the latency will begin to increase. This requires a careful choice of the number of pub-sub nodes in order to achieve high throughput with acceptable latencies. Observe from Figure 33 that the increase in latency due to EventGuard is very small. This is because the wide-area network latencies are of the order of 70ms; while the overhead added at every node by EventGuard is about 2ms. None the less the maximum increase in latency due to EventGuard is lesser than 4%.

**Flooding-based DoS Attacks.** We measured the effect of flooding-based DoS attacks on the throughput of the pub-sub network. We picked one of the leaf nodes to flood the pub-sub network. We vary  $fl$ , the rate that which the malicious node floods messages on the pub-sub network. Figure 34 shows the throughput as  $fl$  increases both in the presence and absence of EventGuard mechanisms to guard the system from flooding-based DoS attacks.

Observe from Figure 34 that in the absence of our guards, the pub-sub system deteriorates drastically with the injection of flooding-based DoS attack. In comparison EventGuard shows a much graceful drop in throughput as the flooding rate  $fl$  increases. Note

that although our guard against flooding-based DoS attacks involves an expensive ElGamal signature check (1.7ms), it restricts the attack into a small neighborhood surrounding the malicious node (see Figure 20). This ensures that the effect of a flooding-based DoS attack is localized and that the rest of the pub-sub network is not affected by it.

### ***3.8 Related Work***

Pub-sub systems can be categorized into two types – *direct channel* and *pub-sub network* depending on the mechanism used for delivering publications from a publisher to relevant subscribers. In a direct channel scheme, a publisher directly delivers a publication to its relevant subscribers. The communication mechanism used in a direct channel could be multiple unicasts or a multicast (if supported by the underlying networking infrastructure). In the absence of wide-area network IP-multicast, the publishers tend to become a performance bottle-neck in the direct channel scheme. In contrast, the pub-sub network delivery model is driven by removing such duplicate messages and performance bottlenecks and improving the system scalability. In a pub-sub network model the publishers and the subscribers communicate via an overlay network (see Figure 18) of nodes connected to one another on top of an existing IP network infrastructure. The publisher sends a publication only to a relatively small subset of nodes in the pub-sub network. The pub-sub network decouples the publishers from the subscribers such that a publisher does not need to be aware of the IP-addresses of all the subscribers. Instead, the pub-sub network is responsible for efficient routing of the publications to the relevant subscribers.

Several pub-sub systems [16, 12, 25] have been developed to provide highly scalable and flexible messaging support for distributed systems. Siena [16] and Gryphon [12] are large pub-sub system capable of content-aware routing. Scribe [25] is an anonymous P2P pub-sub system. Most work on pub-sub systems have focused on performance, scalability and availability. Unfortunately, very little effort has been expended on studying the security aspects of these systems.

It is important to note that both performance and security of pub-sub systems are closely related to the intended usage models. For example, in a scenario where all the subscribers

wish to receive all the event notifications, broadcast is a more economical scheme than content-based pub-sub in terms of messaging cost. Similarly, in a scenario where subscribers can be divided into groups, group addresses and memberships are statically bound, it may be cost-effective to create multicast groups, as each static subset of subscribers is interested in the same notifications. However, when the set of subscribers interested in an event notification is big in size, changes frequently, and geographically disparate, the eligibility of group membership is evaluated dynamically and across multiple administrative domains, a content-based pub-sub overlay service infrastructure is a less costly scheme.

Significant amount of work has been done in the field of secure group communication on multicast networks (survey [75]). Such systems can leverage secure group-based multicast techniques and group key management techniques to provide forward and backward security, scalability and performance. The key problem in such systems arise due to the fact that IP multicast does not provide any mechanisms for preventing non-group members to have access to group communication. A significant restriction with secure group communication is that the group membership has to be predefined. In contrast, EventGuard permits flexible membership at the granularity of subscriptions. Second, EventGuard uses an overlay network and does not rely on IP multicast technology primarily because there have not been Internet scale deployment of the IP multicast protocol.

Wang et al. [109] analyze the security issues and requirements in a content-based pub-sub system. This chapter identifies that the general security needs of a pub-sub application includes confidentiality, integrity and availability. More specifically they identify authentication of publications, integrity of publications, subscription integrity and service integrity as the key issues. The chapter presents a detailed description of these problems in the context of a content-based pub-sub system, but fails to offer any concrete solutions.

Opyrchal and Prakash [66] analyze secure distribution of events in a content-based pub-sub network from a group key management standpoint. They show that previous techniques for dynamic group key management fail in a pub-sub scenario since every event can potentially have a different set of interested subscribers. They use a key caching based technique that relies on subscription popularity to reduce the number of encryptions and to

increase message throughput. However, their approach requires that the pub-sub network nodes (brokers) are completely trustworthy. EventGuard aims to providing security to the subscribers while maintaining confidentiality even from the pub-sub network nodes.

### ***3.9 Summary***

We have presented *EventGuard*, a dependable system architecture for protecting pub-sub services from various attacks. EventGuard offers security features that are critical to pub-sub overlay services, such as authenticity, confidentiality, integrity, and resilience to flooding based DoS attacks. We have described the two key components of EventGuard: The first component is a suite of security guards that secure the basic publish and subscribe operations from DoS attacks and unauthorized reads and writes. These guards can be plugged-into a wide-area content-based pub-sub system in a seamless manner. The second component is a resilient pub-sub network design that is capable of providing secure and yet scalable message routing, countering message dropping-based DoS attacks. A unique feature of EventGuard is its unified security framework that meets both security goal for safeguarding the pub-sub overlay services from various vulnerabilities and threats and performance goal for maintaining the simplicity and scalability of the overall system while providing security guarantees. We have reported a series of experimental evaluations, showing that EventGuard can secure a pub-sub overlay service with minimal performance penalty. Our prototype implementation on top of Siena [16] also demonstrates that EventGuard is easily stackable on any content-based pub-sub core.

## CHAPTER IV

### VOIP NETWORK SECURITY

Peer-to-Peer VoIP (Voice-over-IP) networks, exemplified by Skype, have become increasingly popular because they offer significant cost advantage and richer call forwarding features than traditional public switched telephone networks (PSTN). One of the most important features of a VoIP network is its ability to provide privacy for VoIP clients. The VoIP network uses a peer-to-peer infrastructure (like KaZaa in the case of Skype) to lookup VoIP clients and route voice traffic. This chapter investigates the problem of protecting the identity of a caller from malicious nodes in the VoIP network. We describe two attacks on a VoIP network: a timing analysis attack on the session initiation protocol and a flow analysis attack on the voice session. We have developed solutions to defend a VoIP network against these attacks. We describe an implementation of our caller identification guards as pluggable modules into an open source peer-to-peer Phex client. We present a detailed experimental evaluation to demonstrate the performance and scalability of our guards, while protecting caller identity on a VoIP network.

#### ***4.1 Introduction***

Voice over IP (VoIP), also known as Internet telephony or IP telephony, is emerging as a mature and practical technology alternative to traditional public switched telephone networks (PSTN). VoIP technology enables people to make phone calls through public Internet. As audio quality, bandwidth usage, and setup convenience are reaching acceptable levels, many organizations have switched to VoIP. According to TeleGeography Research [3], world wide VoIP's share of voice traffic has grown from 12.8% in 2003 to an estimated 75% in 2007. One of the most prevailing reasons for such rapid VoIP deployment is reduced costs [4]. Another indisputable reason is the fact that VoIP represents a significant step towards the integration of voice and data networks.

As organizations begin to combine voice and data traffic into a single network, they must ensure manageability, performance, and security, including authorization, authentication, confidentiality and integrity, as well as privacy of phone conversations. Preserving privacy means not only hiding the content of voice traffic, but also hiding who is talking to whom. VoIP security has attracted attentions from both academics and industry in the recent years. Much of the research and development in VoIP security has concentrated on end to end encryption techniques [2]. Such techniques assume that the VoIP network makes it easier to provide anonymity because most VoIP calls are typically made between computers; and these computers have no phone numbers associated with them. Therefore, voice traffic can be protected using end-to-end encryption and routed through low latency anonymizing networks, such as Onion Routing [40], Tor [28], Tarzan [37], or Freedom [11].

Only recently, some researchers have reported vulnerabilities that enables tracking encrypted VoIP flows between two VoIP clients (or so called end point or user agent) [110], and denial of service (DoS) attacks [89]. Surprisingly, very few have studied the potential timing analysis attacks on the network of VoIP relay nodes (proxy nodes) and their detrimental threats to the caller identification problem. This chapter investigates the problem of protecting the identity of a caller (say, its IP-address) from malicious nodes in the VoIP network. For our study, we use the popular Skype [2] VoIP system (used by several millions of users world wide).

The Skype VoIP system uses two main protocols for call control and data delivery. Session Initiation Protocol (SIP) [81] is used to control call setup and termination, while Real-time Transport Protocol (RTP) [88] is used for media delivery. A VoIP call in Skype is processed in two phases. In the lookup phase, the caller searches for the receiver node (identifier by a SIP URL, e.g., `sip:my-db.research.ibm.com`) and sets up a voice path between the caller and the receiver on the VoIP network. The voice path is a sequence of nodes starting with the caller and ending with the receiver such that any two consecutive nodes are neighbors in the VoIP overlay network. In the voice phase, the caller and the receiver send voice packets along the established bi-directional voice path. Once the voice path is set up, low latency anonymizing networks may be used to protect the identity of

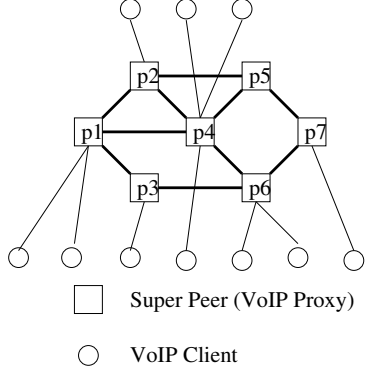
the caller and the receiver in the voice phase [110]. VoIP networks use a peer-to-peer infrastructure (like KaZaa [50] in the context of Skype) to lookup VoIP clients (identified by their SIP URL) on the network. The Skype lookup protocol use a Breath First Search (BFS) based broadcast mechanism to search for a SIP URL on the network and identifies the shortest path on the VoIP network between a caller and a receiver.

## ***4.2 Preliminaries: Skype Lookup Protocol***

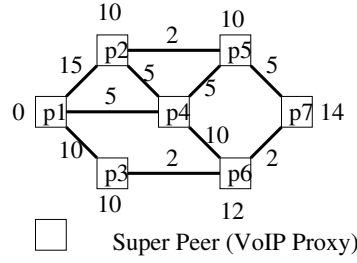
A Skype like VoIP system uses the two main protocols: Session Initiation Protocol (SIP) for call setup and termination, and Real-time Transport Protocol (RTP) for media delivery. As we mentioned in the introduction, a VoIP call in Skype is processed in two phases. In the session initiation phase, the caller searches for the receiver node identified by a SIP URL using a peer-to-peer lookup mechanism and sets up a voice path between the caller and the receiver on the VoIP network. In the voice delivery phase, the caller and the receiver send voice packets along the established bi-directional voice path. SIP architecture uses two basic types of components: SIP clients and SIP proxy servers. Each SIP client is a combination of two entities, the SIP phone and the SIP location server. Each SIP proxy server is either an edge proxy or a relay proxy node. The focus of this chapter is to study the timing attacks on the VoIP flow in the VoIP network rather than the VoIP flow between a SIP client and its edge proxy node.

The Skype VoIP network is based on the KaZaa peer-to-peer technology which employs a super-peer topology to classify peers into two classes: strong peers (VoIP proxies) and weak peers (VoIP clients). Every peer  $p$  in the VoIP network has a set of neighbors  $ngh(p)$  on the Skype overlay network. The VoIP proxies are connected with one another using a power-law like topology [50]. Each of the VoIP clients is connected to exactly one VoIP proxy. Figure 35 illustrates a Skype-like VoIP network topology.

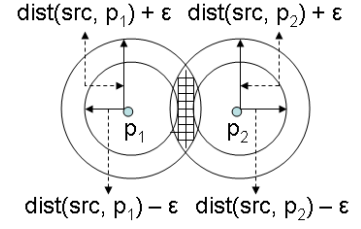
A VoIP call is initiated from a source peer (caller) by issuing a lookup request on the VoIP network using the SIP URL of the destination peer (receiver). The Skype search protocol operates in four steps. First, the `initSearch` initiates a route set up the request from a VoIP client *src*. Second, the `processSearch` processes a search request at some



**Figure 35:** Skype Peer-to-Peer VoIP Network



**Figure 36:** Shortest Path using Skype P2P Lookup Protocol



**Figure 37:** Triangulation Attack Illustration: Caller Lies in Shaded Region

peer on the VoIP network. Third, the `processResult` processes the results of a search request at some peer on the VoIP network. Fourth, the `finSearch` concludes the route set up procedure. We below describe these four operations in detail, which are important for understanding the triangulation based timing attacks to be discussed in the next section.

**initSearch.** A VoIP client  $src$  initiates a search for a receiver  $dst$  by broadcasting  $search(searchId, sipurl = dst.sipurl, ts = curTime)$  to all peers  $p \in ngh(src)$ . The search identifier  $searchId$  is a long randomly chosen unique identifier and  $ts$  denotes the time stamp at which the search request was initiated.

**processSearch.** Let a peer  $p$  receive  $search(searchId, sipurl, ts)$  from its neighbor  $q$ . If peer  $p$  has seen  $searchId$  in the recent past then it drops the search request. Otherwise, peer  $p$  checks if  $sipurl$  is the URL of a VoIP client connected to  $p$ . If yes, peer  $p$  returns its IP address using  $result(searchId, p)$  to peer  $q$ . If not, peer  $p$  broadcasts  $search(searchId, sipurl, ts)$  to all peers  $p' \in ngh(p) - \{q\}$  and caches the search identifier  $\langle searchId, sipurl, q \rangle$  in its recently seen list. Note that  $p'$  has no knowledge of where the search request is initiated.

**processResult.** Let a peer  $p$  receive  $result(searchId, q)$  from peer  $q$ . Peer  $p$  looks up its cache of recently seen search queries to locate  $\langle searchId, sipurl, prev \rangle$ . Peer  $p$  adds a routing entry  $\langle sipurl, q \rangle$  and forwards  $result(searchId, p)$  to peer  $prev$ .

**finSearch.** When the peer  $src$  receives  $result(searchId, q)$  from peer  $q$ , it adds a routing entry  $\langle dst, q \rangle$  to its routing table.

### 4.3 *Attacks on the Session Initiation Protocol*

The first contribution of the chapter is to identify attacks on the lookup phase (SIP) of the VoIP protocols that attempt to compromise caller identity on the VoIP networks. These attacks allow a malicious node on the VoIP network to passively observe the time instants at which they receive search requests and use the VoIP network topology information to determine the caller. We describe three timing attacks on the VoIP network lookup protocol with increasing sophistication: *deterministic triangulation*, *stochastic triangulation* and *differential triangulation*. All of the three attacks exploit the broadcast nature of the search protocol and its shortest path properties to identify the caller with high probability. First, we illustrate triangulation based timing attacks in a simplified setting to highlight the key properties of such attacks. Concretely, we make two assumptions: (i) the network link latencies are deterministic, and (ii) all nodes in the network have a tightly synchronized clock. Second, we relax the first assumption by showing how statistical triangulation attacks can operate on stochastic link latencies (for arbitrary probability distributions) using the notion of stochastic shortest paths. Third, we further relax the second assumption by developing differential analysis techniques to show how differential triangulation attacks can operate solely on relative time instants at which the search request was received by the malicious nodes. The differential triangulation attack can tolerate arbitrary clock skews and is thus agnostic to clock synchronization.

The second main contribution of this chapter is the security mechanisms we have developed to defend against such attacks. Our caller identification guards are designed based on two important observations. First, our triangulation-based attacks have shown that the broadcast nature of the Skype lookup protocol along with its shortest path properties are the primary vulnerabilities that make it susceptible to the caller identification attacks. Second, it is widely acknowledged that a one-way latency of 250ms in a VoIP network is nearly unperceivable to a human user and one-way latency up to 400ms is considered acceptable [95]. Hence, instead of identifying the shortest path between the caller and the receiver, we identify a voice path whose one-way latency is smaller than  $maxLat = 250ms$ . We first

construct a random walk based search algorithm that is resilient to triangulation based timing attacks. However, the voice paths constructed using this lookup algorithm exhibit large path latency ( $\gg maxLat$ ), thereby making it infeasible for practical deployment.

We develop three security guards that alleviate caller identification attacks and yet satisfy the one-way latency constraint of  $maxLat = 250ms$ . Our caller identification guards implement two key ideas. First, we introduce uncertainty into timing information by adding stochastic perturbations to the network latencies. Second, we propose two search algorithms that combine random walk and broadcast algorithms with the goal of reducing lookup latency and yet providing better protection against triangulation based timing attacks. We describe an implementation of our caller identification guards as pluggable modules into an open source peer-to-peer Phex client [1]. We present a detailed experimental evaluation that demonstrates the performance and scalability of our security guards against triangulation based timing attacks in the VoIP proxy networks, while protecting caller identity on a VoIP network.

### 4.3.1 Caller Identification Attacks

The caller identification attacks we consider here are timing analysis based attacks [107]. Timing attacks are serious threats to the low latency network systems [28, 40, 11] and are easy to exploit by well placed malicious attackers [114]. In this section we present three types of triangulation based timing attacks. We assume that the VoIP network topology is publicly known and that some of the network nodes (VoIP proxies) may be malicious. Malicious nodes may collude with one another and use the triangulation based timing attacks described in this section to mount caller identification attacks. In the remaining of this section, we describe the basic model and three representative types of triangulation based timing attacks on the VoIP lookup protocol.

#### 4.3.1.1 *Triangulation Attacks: the Basic Model*

Triangulation based timing attack is a type of timing analysis wherein a malicious node logs the time instants at which it received a search request, find a candidate set of nodes that have a high probability of being the caller, and utilize high correlations to infer the

origin of a call request by aggregating the timing analysis results from multiple malicious nodes. In principle, a triangulation based timing attack operates in three steps: **candidate caller detection**, **candidate caller ranking**, and **triangulation**. In the **candidate caller detection** step, malicious nodes passively observe the time instants at which they receive the search requests. Each malicious node (independently) uses this information and the topology of the VoIP network to deduce a set of candidate callers. In the **candidate caller ranking** step, malicious nodes associate a score with each such candidate caller  $s$  that denotes the likelihood of  $s$  being the actual caller. In the **triangulation step**, two or more malicious nodes combine their sets of candidate callers to obtain a much more *concise* and yet *precise* list of candidate callers.

Concretely, we have identified three triangulation attacks, aiming at compromising the identity of the caller, with increasing sophistication. The first one is called deterministic triangulation attack, which illustrates the effect of the attack using a simplistic model of the VoIP network. We assume that the overlay network latencies are deterministic and that the clocks on all nodes in the network are tightly synchronized. Then we describe the stochastic triangulation attack, which relaxes the first constraint and operates on arbitrary probability distribution functions over the network latencies. Finally, we present the differential triangulation attack, which further removes the tight clock synchronization requirement between all the nodes in the network. It assumes that only the colluding malicious nodes use synchronized clocks, while the clock skew between any two good nodes or a good node and a malicious node can be arbitrarily large.

#### 4.3.1.2 *Deterministic Triangulation Attacks*

By considering triangulation attack in a deterministic latency model of VoIP networks, it allows us to gain a better understanding of both the properties of the Skype lookup protocol and the essence of triangulation inference in the context of VoIP network. In this section, we first use an ideal setup of VoIP network with deterministic latencies to derive some useful properties about the Skype lookup protocol. Then we show how these properties can be exploited to construct caller identification attacks.

### 4.3.1.3 Skype Lookup Protocol Properties

By assuming deterministic latencies of the VoIP network, we observe two important properties of the Skype broadcast based lookup protocol: First, the search protocol identifies the shortest path from a caller proxy  $src$  to a receiver proxy  $dst$ . Second, every node  $p$  that receives the search request knows the length of the shortest path from  $src$  to  $p$ . Formally,

**Lemma 4.3.1** *Let us suppose that the overlay link latencies are deterministic and that all the network nodes have tightly synchronized clocks. Let  $dist(x, y)$  denote the length of the shortest path between nodes  $x$  and  $y$ . The Skype lookup protocol satisfies the following properties:*

- (i) *The protocol establishes the shortest route between the two nodes  $src$  and  $dst$ .*
- (ii) *Any node  $p$  on the network that receives the search request originated from node  $src$  knows  $dist(src, p)$ .*

**Proof** A sketch of the proof is provided here. When a peer  $p$  first receives a search request, that request must have traversed the shortest route between  $src$  and  $p$ . In step `processSearch`, a peer  $p$  can estimate  $dist(src, p)$  using the time stamp  $ts$  on the search request and time instant at which the request was received by peer  $p$ . Also, if the peer  $p$  received the first search request from its neighbor  $q$ , then the shortest route from  $src$  to  $p$  is via  $q$ . Using mathematical induction on the number of hops traversed by a search request, one can show that the route set up step (`processResult`) builds the fastest overlay network path from  $src$  to  $dst$ . ■

Figure 36 illustrates the Skype lookup protocol with  $src = p_1$  and  $dst = p_7$ . The links in the VoIP network are labeled with link latencies (assumed to be deterministic). We label each node with the time instant at which it received the first lookup request starting with peer  $p_1$  at time  $t = 0$ . Evidently, the protocol establishes the shortest route  $p_1 \leftrightarrow p_3 \leftrightarrow p_6 \leftrightarrow p_7$ .

#### 4.3.1.4 Deterministic Shortest Path Triangulation Attack

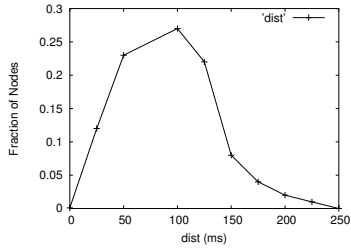
To simplify the discussion, we first discuss the deterministic triangulation attack under the shortest path assumption, that is, a node on the network does not exploit the fact that it may receive multiple copies of a search request. We will discuss multi-path deterministic triangulation attack in the next subsection.

The deterministic shortest path triangulation attack exploits the properties of the Skype lookup protocol and constructs the attack in three steps. The **candidate caller detection** step operates as follows. Let  $p$  be a malicious node that received a search request originating from  $src$  at time  $t = 0$ . The malicious node  $p$  can compute  $dist(s, p)$  for all nodes  $s$  in the network using Dijkstra’s shortest path algorithm in  $O(|E|)$  time (where,  $|E|$  is the number of edges in the network). Given the time instant  $t$  at which the request was received by peer  $p$ ,  $p$  can estimate  $dist(src, p) = t_p$ . Node  $p$  compiles a list of potential callers, denoted by  $S(p)$ , such that for any  $s \in S(p)$ ,  $|dist(s, p) - t_p| < \epsilon$  holds, where  $\epsilon$  is a system supplied control parameter.

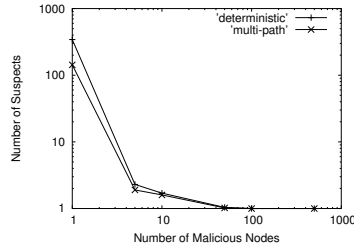
In the **candidate caller ranking** step, the peer  $p$  computes the score for every node  $s \in S(p)$  as  $score_p(s) = \frac{1}{|dist(s, p) - t_p|}$ , and sorts all nodes in  $S(p)$  in the descending order of their scores. The top ranked node  $s$  has the highest likelihood of being the caller.

The **triangulation** step aims at reducing the size of the candidate suspect sets generated by the  $n$  malicious peers ( $n \geq 2$ ). Concretely, a set of colluding malicious nodes, say  $p_1, p_2, \dots, p_n$ , can reduce the number of suspects by computing an intersection (triangulating) over  $S(p_1), S(p_2), \dots, S(p_n)$ . The caller is the node  $s$  that resides in the intersection and has the highest score  $score_{p_i}(s)$  for every  $i \in [1, n]$ . Figure 37 illustrates a triangulation with two malicious nodes  $p_1$  and  $p_2$ . Note that assuming deterministic network latencies makes this attack error free, that is, the caller  $src$  is guaranteed to be a member of the suspect set  $S$ .

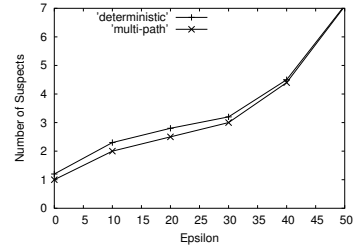
To illustrate the effectiveness of deterministic triangulation attacks under different parameter settings, we use a hypothetically constructed Skype network topology with 1024



**Figure 38:** Distance Distribution



**Figure 39:** Caller Identification with  $\epsilon = 10\text{ms}$



**Figure 40:** Caller Identification with 10 Malicious Nodes

nodes [86]. The topology is constructed using the standard GT-ITM topology generator [125] with node-to-node round trip latencies varying from 24ms-150ms with a mean of 74ms.

Figure 38 shows the distribution of the number of nodes at a distance  $d$  from a randomly chosen node  $p$ . Observe that if  $dist(src, p)$  is either very small or very large then the size of the set  $S(p)$  is likely to be very small. However, without triangulation the attack is largely ineffective in identifying a small and yet accurate suspect set. Figure 39 shows the number of suspects as we vary the number of malicious nodes in the network with  $\epsilon = 10\text{ms}$ . Observe that the number of suspects with one malicious node is 343; with 10 malicious nodes, the number of suspects is reduced drastically to 1.17 (almost uniquely identifying the caller with high probability). This demonstrates the effectiveness of this attack even with a small number of malicious nodes.

Figure 40 shows the number of suspects as we vary the parameter  $\epsilon$  with 10 malicious nodes. A small value for  $\epsilon$  yields the best results. However, when  $\epsilon$  is set too small, even introducing a small uncertainty in the network link latencies (say, small jitters) may result in either an empty suspect set or an incorrect suspect set ( $src \notin S$ ). On the other hand, a large value for  $\epsilon$  identifies a huge candidate set, thereby making the attack less effective. We use the notion of stochastic shortest paths in Section 4.3.1.5 to handle uncertainties in network latencies.

#### 4.3.1.5 Stochastic Triangulation Attack

In this section, we relax the deterministic link latency assumption by modeling the link latencies by independent probability distribution functions. Let us suppose that latency of an edge  $e$  in the network graph is described using a mean  $\mu_e$  and variance  $\sigma_e^2$ . There are several distributions that are exactly described using only mean and variance, including Gaussian, uniform, binomial, exponential distribution, etc. The mean and the variance serve as a good approximation for arbitrary random variables. We extend Dijkstra's shortest path algorithm to operate on stochastic link latencies. The primary idea here is as follows:

- Let  $Z = X + Y$ , where  $X$  and  $Y$  are independent random variables. Then,  $\mu_Z = \mu_X + \mu_Y$  and  $\sigma_Z^2 = \sigma_X^2 + \sigma_Y^2$ .
- We say that the relationship  $(\mu_1, \sigma_1^2) \preceq (\mu_2, \sigma_2^2)$  holds if  $\mu_1 \leq \mu_2 \wedge \sigma_1^2 \leq \sigma_2^2$ , and the relationship  $(\mu_1, \sigma_1^2) \parallel (\mu_2, \sigma_2^2)$  holds if  $\mu_1 \leq \mu_2 \wedge \sigma_1^2 \geq \sigma_2^2$  or  $\mu_2 \leq \mu_1 \wedge \sigma_2^2 \geq \sigma_1^2$ .

Unlike the deterministic setting, the length of a path on the VoIP network is described as a two tuple  $(\mu, \sigma^2)$ , where  $\mu$  is the mean path length and  $\sigma^2$  is the variance of the path length. A path of length  $(\mu_1, \sigma_1^2)$  is *shorter* than a path of length  $(\mu_2, \sigma_2^2)$  if  $(\mu_1, \sigma_1^2) \preceq (\mu_2, \sigma_2^2)$ . A path of length  $(\mu_1, \sigma_1^2)$  is *neither shorter nor longer* than a path of length  $(\mu_2, \sigma_2^2)$  if  $(\mu_1, \sigma_1^2) \parallel (\mu_2, \sigma_2^2)$ . Figure 41 describes a stochastic shortest path algorithm that computes a list of *Pareto-optimal* shortest path lengths from a node  $p$  to all the other nodes on the network. A set of path lengths  $d_1 = (\mu_1, \sigma_1^2), d_2 = (\mu_2, \sigma_2^2), \dots, d_m = (\mu_m, \sigma_m^2)$  is Pareto-optimal if  $d_i \parallel d_j$  for all  $i$  and  $j$  such that  $i \neq j$  and  $1 \leq i, j \leq m$ .

Now we sketch a stochastic triangulation attack in terms of the three step process. Similar to deterministic triangulation attacks, the first step is the **candidate caller detection**, which identifies a candidate suspect set for a given request. Let  $p$  be a malicious node that received a search request from  $src$ . For every node  $v$  in the network,  $p$  computes the shortest stochastic distance  $dist_p[v]$  using the algorithm in Figure 41. Recall that  $dist_p[v]$  is a set of Pareto-optimal stochastic distances between node  $v$  and node  $p$ . The node  $p$  estimates  $dist(src, p) = t_p$  (a scalar), where  $t_p$  denotes the time instant at which node  $p$  received the first search request (assuming the request was initiated by the caller  $src$  at time  $t = 0$ ). A

node  $v$  is added to the suspect set  $S(p)$  if there exists  $d \in \text{dist}_p[v]$  such that  $d = (\mu_d, \sigma_d^2)$  and  $|t_p - \mu_d| < \epsilon$ .

The second step is to compute ranking score for each node  $v \in S(p)$  in terms of the likelihood that  $v$  is the caller. If the link latency distributions were Gaussian, then one can show that the distribution of all distances  $d \in \text{dist}_p[v]$  are Gaussian. Given  $d = (\mu_d, \sigma_d^2) \in \text{dist}_p[v]$ , one can use the Gaussian distribution to determine the likelihood that  $d$  matches the observation  $\text{dist}(\text{src}, p) = t_p$  as  $l_d = \text{Gauss}_{\mu_d, \sigma_d^2}(t_p)$ , where *Gauss* denotes a Gaussian distribution.

For other probability distribution functions, we compute an approximation to this likelihood using Chebyshev inequality:

$$\Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2} \quad (11)$$

Hence, given  $\text{dist}(\text{src}, p) = t_p$  and  $d = (\mu_d, \sigma_d^2) \in \text{dist}_p[v]$ , we determine the likelihood that  $d$  matches  $t_p$  as  $l_d = \frac{1}{k^2}$ , where  $k$  is given by:

$$k = \frac{|t_p - \mu_d|}{\sigma_d}$$

Therefore, one straightforward method to compute this ranking score, denoted by  $\text{score}_p(v)$ , is to use the maximum likelihood  $l_d$  over all  $d \in \text{dist}_p[v]$ .

The third step is to perform the **triangulation** analysis in order to reduce the number of the candidate suspect nodes. Given a set of  $n$  malicious nodes, say  $p_1, p_2, \dots, p_n$ , we compute the aggregate score for a node  $v$  as  $\text{score}(v) = \frac{\sum_{i=1}^n \text{score}_{p_i}(v)}{n}$ . Now, the adversary sorts all the nodes in the descending order of their aggregated scores. The caller is very likely to have a high score and thus appear within the top few entries.

Figure 42 shows that the stochastic triangulation attack is more effective than a deterministic triangulation attack when there are uncertainties in link latencies. We assume that there are ten malicious nodes in the network. The figure shows the probability that the caller appears in the top-10 entries using a Gaussian distribution for link latencies ( $\sigma_e \cdot \mu_e$  on x-axis). The figure shows the results for the deterministic triangulation attack using the best setting for the parameter  $\epsilon$ . Note that initially as  $\epsilon$  increases the probability of

```

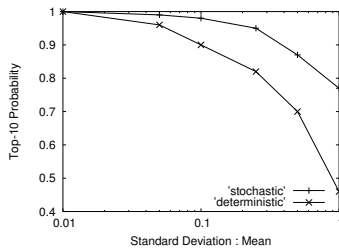
STOCHASTIC SHORTEST PATH(Graph  $G$ , Peer  $p$ )
(1) for each vertex  $v \in V(G)$ 
(2)    $dist[v] = \{(\infty, \infty)\}$ 
(3)    $label[v] = \text{false}$ 
(4) end for
(5)  $dist[p] = (0, 0)$ 
(6)  $label[p] = \text{true}$ 
(7) while pick a labeled vertex  $v$ 
(8)    $label[v] = \text{false}$ 
(9)   for each neighbor  $u \in ngh(v)$ 
(10)     $label[u] = \text{true}$ 
(11)    for each distance  $(\mu_v, \sigma_v^2) \in dist[v]$ 
(12)     let  $d = (\mu_v + \mu_{evu}, \sigma_v^2 + \sigma_{evu}^2)$ 
(13)     remove any  $d' \in dist[u]$  if  $d \preceq d'$ 
(14)     add  $d$  to set  $dist[u]$  if for every  $d' \in$ 
       $dist[u], d \parallel d'$ 
(15)    end for
(16)  end for
(17) end while

```

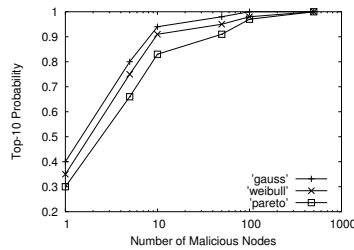
**Figure 41:** Stochastic Shortest Path Algorithm

the deterministic triangulation attack increases; however, after a critical value, increasing  $\epsilon$  decreases the effectiveness of the attack.

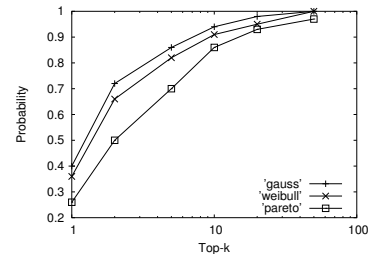
Figure 43 shows the probability that the caller appears in the top-10 entries in the ranked list for three different link latency distributions: Gaussian, Weibull and Pareto distributions [82], by varying the number of malicious nodes. We set the  $\sigma_e:\mu_e = 0.25$ . Even for a small number of malicious nodes increases, the probability of a successful attack increases significantly. The attack is most effective against Gaussian distribution primarily because



**Figure 42:** Stochastic Vs Deterministic Triangulation Attack



**Figure 43:** Top-10 Probability



**Figure 44:** 10 Malicious Nodes

it does not use Chebyshev inequality to compute an approximate score. Pareto distribution reduces the efficacy of the attacks the most because of its higher order moments [83] are larger than the Weibull distribution. Thus, the Chebyshev inequality based approximation for the Weibull distribution is better than that for the Pareto distribution.

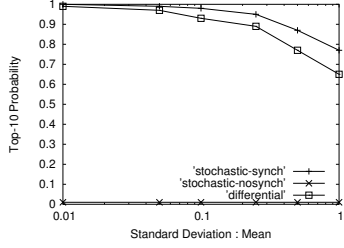
Figure 44 shows the probability that the caller appears in the top- $\kappa$  entries in the ranked list for varying  $\kappa$  under three types of link latency distributions. There is more than a 25% chance that the caller is the top most entry in the suspect list, thereby making the attack highly effective. Similar to Figure 43 the attack is most effective against a Gaussian distribution and least effective against a Pareto distribution.

#### 4.3.1.6 Differential Triangulation Attack

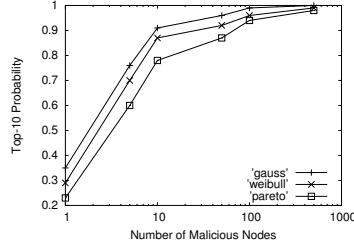
The differential triangulation attack removes the tight clock synchronization requirement between all the nodes in the network. It requires that only the colluding malicious nodes use synchronized clocks. Thus, the time stamp  $ts$  in the search request could be some randomly generated number thereby making it hard for a malicious node  $p$  to estimate  $dist(src, p)$ . The key idea behind differential triangulation attack is that, two colluding malicious nodes  $p$  and  $q$  can estimate the difference  $dist(src, p) - dist(src, q) = t_p - t_q$  based on the time instants ( $t_p$  and  $t_q$ ) at which the first broadcast packet was received by the nodes  $p$  and  $q$ .

The differential triangulation attack follows the same steps of a stochastic triangulation attack and estimates the stochastic distance of every node  $v$  from node  $p$ :  $dist_p[v]$ . Recall that  $dist_p[v]$  is a set of Pareto-optimal stochastic shortest path lengths from node  $v$  to node  $p$ . We approximate  $dist_p[v]$  by computing a mean over all stochastic distances  $d \in dist_p[v]$  as  $\widehat{dist}_p[v] = (\frac{1}{d} * \sum_{d \in dist_p[v]} \mu_d, \frac{1}{d^2} * \sum_{d \in dist_p[v]} \sigma_d^2)$ . One can now compute the stochastic distance  $dist_{pq}[v] = \widehat{dist}_p[v] - \widehat{dist}_q[v]$  as  $(\mu_p[v] - \mu_q[v], \sigma_p^2[v] + \sigma_q^2[v])$ , where  $\widehat{dist}_p[v] = (\mu_p[v], \sigma_p^2[v])$  and  $\widehat{dist}_q[v] = (\mu_q[v], \sigma_q^2[v])$ . Finally, we add a node  $v$  to the candidate caller set if  $|(t_p - t_q) - (\mu_p[v] - \mu_q[v])| < \epsilon$ , where  $\epsilon$  is a system supplied parameter.

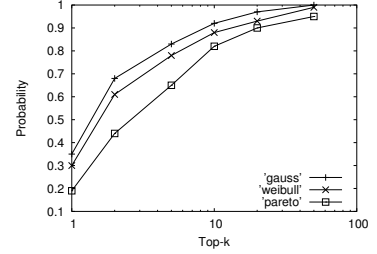
We assign a score to a node  $v$  ( $score_{pq}(v)$ ) based on the likelihood that the stochastic distance  $dist_{pq}[v]$  matches the observation  $t_p - t_q$ . The likelihood computations are similar to that used in the stochastic triangulation attack. We use the exact likelihood computation if



**Figure 45:** Differential Vs Stochastic Triangulation Attack



**Figure 46:** Top-10 Probability



**Figure 47:** 10 Malicious Nodes

the link latency distribution is Gaussian  $score_{pq}(v) = Gauss_{\mu_p[v]-\mu_q[v], \sigma_p^2[v]+\sigma_q^2[v]}(t_p - t_q)$ . For all other distributions we use an approximation based on Chebyshev inequality  $score_{pq}(v) = \frac{1}{k^2}$ , where  $k = \frac{(t_p - t_q) - (\mu_p[v] - \mu_q[v])}{\sqrt{\sigma_p^2[v] + \sigma_q^2[v]}}$ .

When there are more than two colluding malicious nodes  $p_1, p_2, \dots, p_n$ , the triangulation step operates as follows. First, we use one arbitrarily chosen malicious node (say  $p_1$ ) as the reference node. We compute  $score_{p_i p_1}(v)$  for all nodes  $v$  and  $i > 1$ . We compute the average score for a node  $v$  as  $\frac{\sum_{i=2}^n score_{p_i p_1}(v)}{n-1}$ . The nodes are sorted in decreasing order of their scores. Similar to stochastic triangulation attack, the caller is likely to have a high score and thus appear within the top few entries.

Figure 45 compares the differential triangulation attack against the stochastic triangulation attack. Assuming that the clock on all nodes is synchronized, the stochastic triangulation attack performs slightly better. On the other hand, the stochastic and the deterministic triangulation attacks are completely ineffective when the clocks are out of synch. Figures 46 and 47 shows the probability of a successful attack using three types of link latency distributions. These results are similar to that for stochastic triangulation attack, albeit a small decrease in the probability of success. Nonetheless, the differential triangulation attack can operate even when the time stamp  $ts$  is substituted with some randomly generated number.

### 4.3.2 Countering Triangulation based Timing Attacks

#### 4.3.2.1 Design Overview

A careful study of triangulation timing attacks leads us to believe that the shortest path nature of the broadcast search algorithm is a fundamental source that gives the attackers

an opportunity to mount the caller identification attacks. Even though we have identified deterministic, stochastic, and differential triangulation based timing attacks, we conjecture that other types of statistical inference attacks on distributed shortest path algorithms may exist. Thus solutions to countering statistical timing attacks should fundamentally change the search algorithm by introducing uncertainty into the search protocol.

Interestingly, it is widely acknowledged that a one-way latency of 250ms is nearly unperceivable to a human user and one-way latency up to 400ms is considered acceptable [95] in all VoIP systems in practice. In fact, these are the baselines for setting the system-supplied timeout for messages on the VoIP networks. Therefore, in a VoIP network one does not need to identify the shortest path between the caller and the receiver. We exploit this fact to devise a more secure and yet efficient search protocol that can identify a voice path while preserving real-time latency constraint.

One approach to countering caller identification attacks is to add random noise to both link latencies and the VoIP lookup algorithm. We can introduce random noise into link latencies in the Skype lookup protocol by having each node add some stochastic perturbation before broadcasting a search request to all its neighbors. The first mechanism implemented in our caller identification guard is to add a Gaussian noise  $(\mu_{noise}, \sigma_{noise}^2)$  with a low mean and high variance. A low  $\mu_{noise}$  ensures that the latency of the VoIP path is near optimal. A high  $\sigma_{noise}^2$  adds uncertainty into triangulation based timing analysis, thereby reducing its effectiveness. Nonetheless, this is equivalent to replacing the latency of each edge  $e$  described by link latency  $(\mu_e, \sigma_e^2)$  with a perturbed latency  $(\mu_e + \mu_{noise}, \sigma_e^2 + \sigma_{noise}^2)$ . Clearly, the stochastic and differential analysis attacks apply even in the presence of latency perturbations.

Another approach is to use a random walk based search algorithm that is resilient to triangulation based timing attack. However, the random walk approach may set up highly sub-optimal voice paths, thereby violating the one-way latency constraint. To guarantee the real-time property and yet provide resilience to triangulation based timing attacks, we propose a deferred shortest path broadcast search algorithm with two control knobs:  $\gamma$  for controlling the random walk path length and  $\omega$  for controlling the number of random

walkers a node may support. By carefully tuning these two control knobs, we can provide an effective combination of the random walk and the broadcast algorithm with the goal of reducing lookup latency and yet providing better protection against caller identification attacks.

In the rest of this section, we first discuss the random walk search algorithm and study its lookup efficiency and its resilience to caller identification attacks. Then, we present our deferred shortest path broadcast algorithm that carefully combines the random walk search with broadcast search. We report our experimental evaluation results and demonstrate the effectiveness of these algorithms against triangulation based timing attacks described in Section 4.3.

#### 4.3.2.2 *Random Walk Search Algorithm*

Similar to the Skype protocol the random walk search algorithm operates in four steps.

**initSearch.** A VoIP client  $src$  initiates a search for a receiver  $dst$  (identified by its SIP URL) by sending  $search(searchId, sipurl = dst.sipurl)$  to a uniformly and randomly chosen neighbor  $q \in ngh(src)$ . The search identifier  $searchId$  is a long randomly chosen unique identifier.

**processSearch.** Let a peer  $p$  receive  $search(searchId, sipurl)$  from its neighbor  $q$ . Peer  $p$  checks if  $sipurl$  is the URL of a VoIP client connected to  $p$ . If yes, peer  $p$  returns its IP address  $result(searchId, p)$  to peer  $q$ . If not, peer  $p$  uniformly and randomly chooses a neighbor  $q \in ngh(p)$ . The peer  $p$  sends  $search(searchId, sipurl)$  to all peers  $q$ . If the peer  $p$  has not previously seen the search identifier  $searchId$ , it caches  $\langle searchId, sipurl, q \rangle$  in its recently seen list.

**processResult.** Let a peer  $p$  receive  $result(searchId, q)$  from peer  $q$ . Peer  $p$  looks up its cache of recently seen search queries to locate  $\langle searchId, sipurl, prev \rangle$ . Peer  $p$  adds a routing entry  $\langle sipurl, q \rangle$  and forwards  $result(searchId, prev)$  to peer  $prev$ .

**finSearch.** When the peer  $src$  receives  $result(searchId, q)$  from peer  $q$ , it adds a routing entry  $\langle dst, q \rangle$  to its routing table.

There are two interesting properties of the random walk based search algorithm that makes it resilient to triangulation based timing attacks. First, the Markovian property (memoryless) of the random walk algorithm makes it resilient to triangulation based timing attacks. Let us suppose that a random walker visits two malicious nodes  $p_1$  and  $p_2$  in time  $t_1$  and  $t_2$  (respectively) with  $t_1 < t_2$ . The Markovian property of the random walk algorithm ensures that the time instants at which the request reaches  $p_2$  nodes do not leak any additional information to the adversary. Formally, given the fact that a random walker visited node  $p_1$  first, the probability that it would visit node  $p_2$  in the future is independent of the caller  $src$ .

$$Pr(\text{visit } p_2 \mid \text{visit } p_1 \wedge \text{caller} = src) = Pr(\text{visit } p_2 \mid \text{visit } p_1)$$

Hence, neither the time instant  $t_2$  or the difference  $t_2 - t_1$  provides any additional information to the adversary. In fact, the above argument holds for any number malicious nodes visited by the random walker. Hence, when a random walker visits a set of malicious nodes  $p_1, p_2, \dots, p_n$  (in that order), the only useful information is the fact that the random walker first visited node  $p_1$ . Formally,

$$Pr(src = v \mid \text{visit } p_1, p_2, \dots, p_n) = Pr(src = v \mid \text{visit } p_1 \text{ first})$$

Second, the random walk algorithm is not vulnerable to shortest path based timing attacks because a random walker does not essentially traverse the shortest path between any two nodes. We use *random walk distance* based timing attacks to study the resilience of the random walk algorithm. We define random walk distance between two nodes  $u$  and  $v$  ( $rwdist_u[v]$ ) as the expected number of hops required for a random walker to reach node  $v$  starting from node  $u$ . Unlike the shortest path distance, the random walk distance between two nodes  $u$  and  $v$  may not be symmetric, that is,  $rwdist_u[v] \neq rwdist_v[u]$ .

Let  $p$  be the first malicious node that received a search request originating from caller  $src$ . If there exists no such node  $p$  then the search was completely secure. For every node  $v$ , we compute the probability a random walk starting from node  $v$  reaches malicious node  $p$  before it reaches any other malicious node in the network. Using standard Markov model

theory, we model the random walk using a transition probability matrix  $M$ . An entry  $M_{ij} = \frac{1}{|ngh(i)|}$  if  $j \in ngh(i)$ ; 0 otherwise, where  $ngh(i)$  denotes the set of neighbors of peer  $i$ .  $M_{ij}$  denotes the probability that a random walker will visit node  $j$  given the walker is currently in node  $i$ . We determine the probability that a random walker starting from node  $v$  reaches node  $u$  in step  $d$   $pr_v^d[u]$  from  $\pi M^d$ , where  $\pi$  is a row vector with  $\pi[i] = 1$  if  $i = v$ ; 0 otherwise. Every malicious node  $p$  computes the random walk distance from a node  $v$  as  $rwdist_v[p]$ :

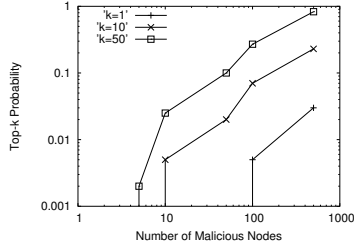
$$rwdist_v[p] = \sum_{d=1}^{\infty} d * pr_v^d[p] * \left( \prod_{i=1}^{d-1} (1 - pr_v^i[p]) \right) \quad (12)$$

While theoretically the summation extends up to infinity, in practice  $pr_v^d[p]$  tends to converge to a steady state value for all  $d > O(N \log N)$ , where  $N$  is the number of nodes in the network.

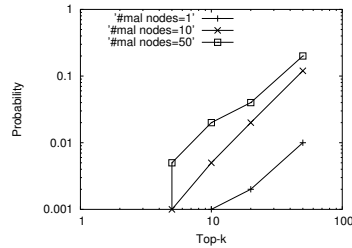
A set of colluding malicious nodes  $p_1, p_2, \dots, p_n$  where  $p_1$  received the first random walk request operate as follows. For every node  $v$  the adversary compiles  $rwdist_v[p_i]$  for all  $1 \leq i \leq n$  and sort them in increasing order. Let us suppose that  $p_1$  is the  $\eta^{th}$  smallest element in the sorted list. Then, we associate a score with node  $v$  as  $score_{p_1}[v] = \eta$ . We sort the nodes by their score as follows:  $v_1 \prec v_2 := (score_{p_1}[v_1] < score_{p_1}[v_2]) \vee (score_{p_1}[v_1] = score_{p_1}[v_2] \wedge rwdist_{v_1}[p_1] < rwdist_{v_2}[p_1])$ . Similar to the triangulation based timing attacks, the true caller is more likely to appear in the top few entries of this sorted list.

Figure 48 shows the efficacy of caller identification as the number of malicious nodes increases. Figure 49 shows the probability that the actual caller appears in the top- $\kappa$  nodes of the sorted list. Even though the random walk algorithm is susceptible to caller identification attacks, the probability of success in such an attack is significantly smaller than that of triangulation based attacks on the Skype broadcast search protocol (see Section 4.3). For example, when the VoIP network has 10 malicious nodes, the probability that a caller appears in the top-10 list is 0.05 using the random walk algorithm and 0.75 using the broadcast search algorithm.

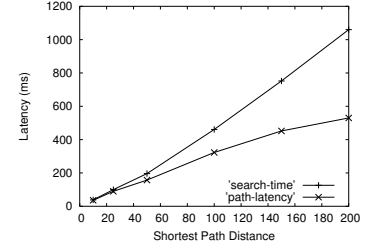
Figure 50 shows the search time and the one-way latency of the paths set up by a random walk search algorithm. The major drawback with the random walk search algorithm is its enormous search time and grossly sub-optimal VoIP paths, thereby making the



**Figure 48:** Top- $\kappa$  Probability



**Figure 49:** Number of Malicious Nodes



**Figure 50:** Random Walk Search Algorithm

approach infeasible for practical deployment. In the following sections, we present hybrid techniques that attempt to combine triangulation attack resilience from the random walk search algorithm and optimal (shortest) path setup using the broadcast search algorithm.

#### 4.3.2.3 Deferred Shortest Path Broadcast Algorithms

In this section, we propose two hybrid techniques that combine random walk and broadcast algorithms with the goal of reducing lookup latency and yet providing good protection against caller identification attacks. We present detailed experimental results on these algorithms in Section 4.3.3.

#### $\gamma$ -Controlled Random Walk

In this section, we propose a controlled combination of the random walk search algorithm and the broadcast search algorithm. We use a global system wide parameter  $\gamma$  which limits the length of random walk. In this protocol, the search algorithm operates in two phases: random walk search phase (RW) and broadcast search phase (B). The algorithm starts operating at node  $src$  in phase RW. In phase RW, when a node  $p$  receives a search request, with probability  $\gamma$  it continues to operate using the random walk search algorithm; with probability  $1 - \gamma$  the algorithm changes to phase B. In phase B, it uses the Skype broadcast search algorithm to broadcast the search request. Once the request enters phase B, it continues to operate in that phase.

This algorithm ensures that the average number of hops in the random walk phase is  $\frac{1}{1-\gamma}$  and the probability that the length of the random walk exceeds  $d$  hops is  $\gamma^d$ . Starting at node  $src$ , let us suppose that at the end of phase RW the request reaches node  $q$ . The

broadcast algorithm identifies the shortest path between node  $q$  to node  $dst$ . Evidently, a small value for  $\gamma$  ensures that the latency of the VoIP path is near optimal.

A triangulation based timing attack may identify  $q$  (broadcast initiator) with high probability. However, identifying the caller  $src$  would be non-trivial to the attackers. Let the earliest malicious nodes that participated in phase RW be node  $p$ . If there exists no such malicious node then we set  $p$  to the broadcast initiator  $q$ . Hence, node  $p$  is the first node known to the adversary that received the random walk search request. Now, one can use a similar statistical inference attack described in Section 4.3.2.2 to determine the caller. The key difference in the inference attack is that the parameter  $\gamma$  reduces the probability that a random walker starting from node  $v$  reaches node  $u$  in  $d$  hops ( $pr_v^d[u]$ ) by a factor  $\gamma^d$  (in Equation 12). Clearly, a small value for  $\gamma$  indicates that the actual caller is close to node  $p$  and thus improves the efficacy of a statistical inference attack.

#### **$\omega$ -Controlled Multi-Agent Random Walk**

The second solution is to use a multi-agent random walk search algorithm. This solution is very similar to the random walk search algorithm except that the caller  $src$  sends out  $\omega$  random walkers. Sending out a large number of random walkers reduces the search time and helps  $src$  discover a shorter path to node  $dst$ . Clearly, as  $\omega$  increases the expected one-way latency of the VoIP path decreases. Indeed one can show that as  $\omega$  tends to infinity, the path latency asymptotically decreases to the shortest path.

However, if all the random walkers were sent out by  $src$  at time  $t = 0$ , then this algorithm is vulnerable to triangulation based timing attack. Let  $p(rw)$  denote the first malicious node visited by random walker  $rw$ . The key idea here is that two colluding malicious nodes  $p(rw_1)$  and  $p(rw_2)$  can estimate  $rwdist_{src}[p(rw_1)] - rwdist_{src}[p(rw_2)]$  when receive their first random walk request from two different random walkers  $rw_1$  and  $rw_2$ . As discussed earlier, if the same random walker  $rw$  visits both  $p_1$  and  $p_2$  (in that order), then the adversary does not gain any additional information from  $p_2$  because of the Markovian property of the random walk algorithm. Given an estimate of  $rwdist_{src}[p(rw_1)] - rwdist_{src}[p(rw_2)]$ , this attack is very similar to the differential triangulation attack wherein we use random walk distance  $rwdist$  instead of the shortest distance  $dist$  (see Section 4.3).

Obviously, *rwdist* has more uncertainty built into it because of the probabilistic nature of the random walk search algorithm; thus a triangulation attack on *rwdist* is likely to be less effective than a triangulation attack on *dist*. As the number of random walkers  $\omega$  increases, we may have  $\omega$  malicious nodes  $p(rw_i)$  ( $1 \leq i \leq \omega$ ) such that random walker  $rw_i$  visited malicious node  $p(rw_i)$  first. Hence, a large  $\omega$  improves the efficacy of the triangulation based timing attack.

Fortunately, one can mitigate such triangulation based timing attacks using concurrent search requests. The key idea is to ensure that a malicious node cannot associate two random walkers with the same search request. One simplistic approach is to assign a different *searchId* for each of the  $\omega$  random walkers; however, since they carry the same destination *sipurl*, the malicious nodes may temporally correlate all requests on the same *sipurl*. We use a solution proposed by Song et al. [94] for searches on encrypted data to ensure that the *sipurl* is not revealed to none other to peer *dst* on the VoIP network. The peer *src* replaces *searchId* and *sipurl* in the search request by  $\langle searchId, F_{searchId}(sipurl) \rangle$ , where  $F$  is a keyed pseudo-random function (PRF) and *searchId* is a long randomly chosen nonce for a random walker. A peer  $p$  on the VoIP network checks whether a search request for  $\langle searchId, match \rangle$  matches *sipurl* by checking if  $F_{searchId}(sipurl) = match$ . One can show that a peer  $p$  who does not know *sipurl* does not obtain any additional information from the search request. For a detailed proof refer to [94].

### 4.3.3 Experimental Evaluation

#### 4.3.3.1 Implementation Sketch

In this section, we describe a brief implementation of our algorithms using Phex [1]: an open source Java based implementation of peer-to-peer broadcast search protocol. A VoIP protocol like Skype operates on top of the peer-to-peer infrastructure. We have implemented our algorithms as pluggable modules that can be weaved into the Phex client code using AspectJ [33]. Our implementation is completely transparent to the VoIP protocol that operates on top of the peer-to-peer infrastructure. Also, our implementation does not require any changes to topology construction and maintenance algorithms (as nodes join,

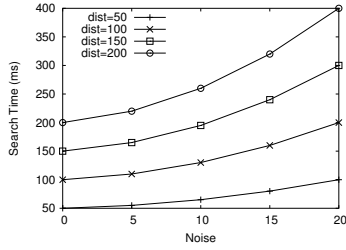
leave, fail or recover) and the underlying TCP/IP or UDP based communication libraries.

Below we sketch our implementation of three algorithms: latency perturbation, controlled random walk and multi-agent random walk. As described in Section 4.2, a broadcast search protocol has four parts: `initSearch`, `processSearch`, `processResult` and `finSearch`. Our algorithms require changes only to the `processSearch` part. This part is implemented in Phex using several methods of which we are interested in only the following: `receiveRequest`, `checkDuplicate`, `matchQuery`, and `requestForward`.

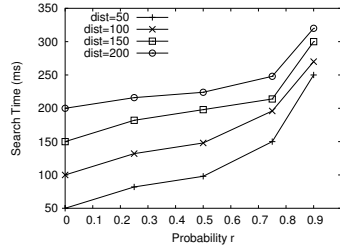
Latency perturbation requires that we add  $(\mu_{noise}, \sigma_{noise}^2)$  for every edge on the network. This is accomplished by intercepting an outgoing search request at a Phex client using the `requestForward` method. We buffer and delay the request for  $delay \sim Gauss_{\mu_{noise}, \sigma_{noise}^2}$  time units before processing the request. The random walk search algorithm requires a request to be processed even if its `searchId` has been seen in the past. We bypass the call from the `receiveRequest` method to `duplicateCheck` method when using the random walk search algorithm. Note that processing a request and setting up VoIP path is identical for both the controlled random walk algorithm and the broadcast search algorithm. However, for the multi-agent random walk algorithm we intercept the method `matchQuery` and substitute a simple hash table based matching operation to a PRF based matching operation (see Section 4.3.2). Finally, we need to change the request forwarding step. We intercept request forwarding using the `requestForward` method. In the broadcast algorithm the `requestForward` method on node  $p$  returns all the nodes in  $ngh(p)$ ; instead, we return only one randomly chosen neighbor.

#### 4.3.3.2 Experimental Results

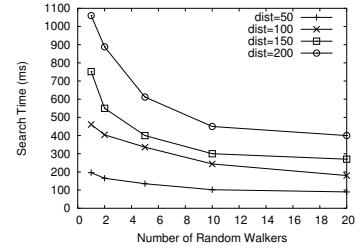
In this section we present experimental results on our algorithms for mitigating caller identification attacks: latency perturbation, controlled random walk and multi-agent random walk. Our experiments are divided into two parts. The first part measure the performance of these algorithms using two metrics: search time and path latency. These performance numbers are intrinsically related to the parameters used by these algorithms:  $(\mu_{noise}, \sigma_{noise}^2)$  for latency perturbation, the probability of random walk ( $\gamma$ ) for controlled random walk,



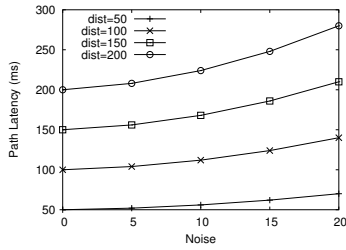
**Figure 51:** Latency Perturbation



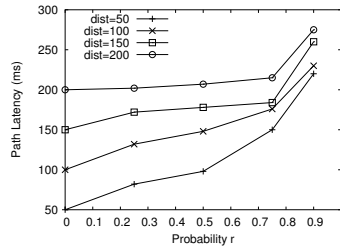
**Figure 52:** Controlled Random Walk



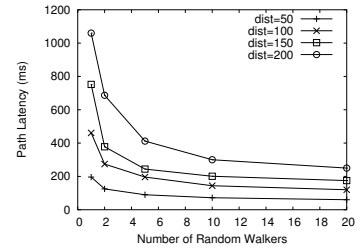
**Figure 53:** Multi-Agent Random Walk



**Figure 54:** Latency Perturbation



**Figure 55:** Controlled Random Walk



**Figure 56:** Multi-Agent Random Walk

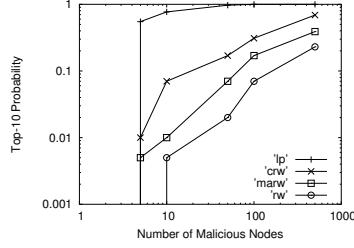
and the number of random walkers ( $\omega$ ) for multi-agent random walk. Using the performance results we determine parameter settings for these algorithms such that 99% of VoIP calls have a one-way path latency smaller than  $maxLat = 250ms$ . The second part of our evaluation uses these parameter settings to evaluate the efficacy of these algorithms in defending against caller identification attacks. We use the probability of a successful attack as the metric for measuring the effectiveness of the attack. We study this probability as the number of malicious nodes varies in the network.

#### 4.3.3.3 Performance Results

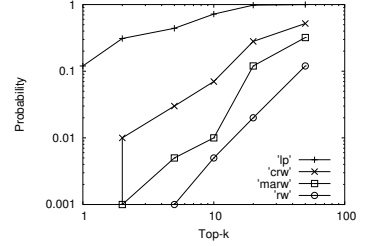
Figures 51, 52 and 53 show the search time under various parameter settings for our caller identity protection algorithms described in Section 4.3.2. Search time determines the time period between a caller initiating a VoIP call and the establishment of a voice session between the caller and the receiver. Larger search time causes an initial delay in session set up but does not affect the quality of the voice conversation. Figures 54, 55 and 56 show the path latency under various parameter settings of our caller identity protection algorithms. As described earlier, path latencies below 250ms is unperceivable to human

	99%	95%	90%
$\sigma_{noise}$	14	15	20
$\gamma$	0.76	0.79	0.83
$\omega$	20	18	12

**Figure 57:** Optimal Parameter Setting



**Figure 58:** Top-10 Probability



**Figure 59:** Top-k

beings. However, higher path latencies are known to significantly deteriorate the quality of voice conversations.

#### 4.3.3.4 Optimal Parameter Setting

We randomly chose 1024 pairs of callers and receivers. We vary algorithm parameters: increase  $\sigma_{noise}^2$ ,  $\gamma$  and decrease  $w$  until  $X\%$  of the pairs have a path latency under  $maxLat$ . We use a binary search strategy to identify the optimal parameter values. For instance, we determine  $\sigma_{noise}^2$  that satisfies  $X\%$  latency constraint as follows. We start with a range  $(0, 100)$ , where  $\sigma_{noise} = 0$  satisfies the  $X\%$  latency constraint, while  $\sigma_{noise} = 100$  does not satisfy the constraint. Given a range  $(l, u)$  we experiment with  $\sigma_{noise}$  set to  $\frac{l+u}{2}$ . If  $\sigma_{noise} = \frac{l+u}{2}$  satisfies  $X\%$  latency constraint then the new range is set to  $(\frac{l+u}{2}, u)$ ; otherwise the new range is set to  $(l, \frac{l+u}{2})$ . We repeat this binary search until the size of the range  $(u - l)$  is acceptably small. When the search terminates, we have the optimal parameter setting  $\sigma_{noise}^{opt} = l$ .

Figure 57 summarizes our parameter settings for different percentile latency constraints. Our goal is to study the resilience of these algorithms under the constraint that the path latency is smaller than  $maxLat$ . Note that the resilience to caller identification attack monotonically increases with  $\sigma_{noise}^2$  and  $\gamma$ ; and monotonically decreases with  $\omega$ . Hence, studying attack resilience of these algorithms under the parameter setting in figure 57 is sufficient.

#### 4.3.3.5 *Attack Resilience*

In this section we compare the attack resilience of our algorithms: latency perturbation (lp), controlled random walk (crw) and multi-agent random walk (marw). We use  $\omega = 1$  random walker for base line comparison (rw). Giving at most importance to the quality of the voice session, we use the 99% optimal parameter setting from figure 57 in this experiment. Of course, one should keep in mind that the random walk search algorithm (rw) violates the latency constraint. Figure 58 shows the probability that the actual caller appears in the top-10 nodes of the sorted list as the number of malicious nodes increases using different attack resilient algorithms. Figure 59 shows the probability that the actual caller appears in the top- $\kappa$  nodes of the sorted list with 10 malicious nodes in the network using different attack resilient algorithms. These figures indicate that the latency perturbation does not offer much resilience to caller identification attacks. The multi-agent random walk algorithm performs the best though not matching the  $\omega = 1$  random walk algorithm. Recall that unlike the random walk algorithm, the multi-agent random walk algorithm is vulnerable to triangulation attacks on random walk distances.

As discussed in Section 4.3.2 the resilience of multi-agent random walk algorithm improves as the number of concurrent searches increases. Note that concurrent searches make it hard for the attackers to associate two random walkers with the same caller. Figures 58 and 59 study multi-agent random walk assuming there are no concurrent searches. A detailed analysis on the multi-agent random walk algorithm with concurrent searches is outside the scope of this chapter.

### 4.4 *Attacks on Voice Session*

In the first half of this section, we focus on flow analysis attacks on VoIP networks. A flow (sufficiently long lasting packet stream) analysis attack measures the volume of flow (packets per unit time) between two nodes on the VoIP network and uses this information in conjunction with the VoIP network topology to determine a caller & recipient pair. Figure 60 shows the traffic volume between two Skype peers  $a$  and  $b$  as the number of VoIP flows between  $a$  and  $b$  increases. The constant packet rate of VoIP flows makes it easier for an

external observer to trace a flow from a caller to a receiver. Figure 61 presents a simple scenario with two callers ( $s_1$  and  $s_2$ ) and two receivers ( $d_1$  and  $d_2$ ). Evidently an adversary can track the volume of VoIP flow from the caller  $s_1$  to node  $p_2$  to node  $p_5$  to the recipient  $d_1$ . We describe three statistical inference based flow analysis attacks on VoIP networks with increasing sophistication. These attacks help an adversary to identify a small list of potential recipients for a given VoIP call.

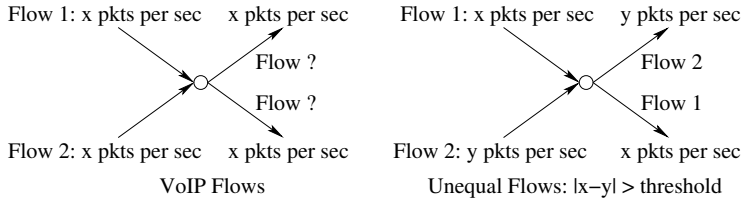
In the second half of this section, we develop practical techniques to achieve quantifiable and customizable privacy on VoIP networks. We reduce the efficacy of flow analysis attacks by mixing one or more VoIP flows. The constant packet rate nature of VoIP places makes it easy for one or more VoIP flows to be mixed without leaking much information to an external observer. Figure 62 shows a simple example mixing two VoIP flows as against mixing two flows with different packet rates. Indeed an external observer can infer nothing more than the number of VoIP flows between two Skype peers  $a$  and  $b$ . Figure 63 illustrates this approach using the two caller and two receiver scenario in Figure 61. The node  $p_4$  mixes the two VoIP flows thereby making it hard for the adversary to determine the recipient for a call originating from caller  $s_1$ . Observe that this provides  $k = 2$  anonymity for both the voice calls. Intuitively, if a VoIP flow from  $src$  to  $dst$  is routed through a Skype peer that processes a large number of flows then the caller anonymity set is likely to be large.

We propose an anonymity-aware session initiation protocol (AASIP). The AASIP protocol allows a caller  $src$  to specify a personalized and customizable anonymity parameter  $k$  for every VoIP call. The AASIP protocol intelligently sets up a route for a VoIP flow from  $src$  to  $dst$  on the peer-to-peer VoIP network such that the recipient anonymity set (number of possible recipients inferred by the adversary) is at least as large as  $k$ . There are at least two important challenges in designing the AASIP protocol. First, the real time nature of voice conversations mandate that the end-to-end one-way latency is no larger than a threshold  $maxLat$ ; typically, delays up to 250ms is unperceivable to a human user, while delays up to 400ms is considered acceptable [95]. Second, the resource constraints on a Skype peer limit the total number of VoIP flows  $maxFlow$  that can be routed through it.

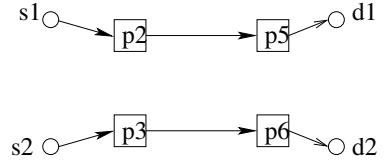
In the final portion of this section, we discuss potential attacks on the AASIP protocol

Num VoIP Flows	1	4	16	64
IP Bandwidth (Kbps)	24	96	384	1536

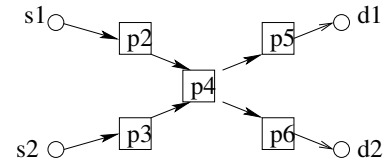
**Figure 60:** VoIP Flow Bandwidth: G.729A (CS-CELP) Audio Codec with 8 Kbps Compression and Packet Duration 20ms



**Figure 62:** Mixing Flows



**Figure 61:** Flow Analysis Attack

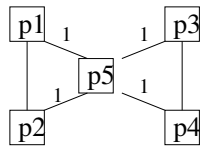


**Figure 63:** Resisting Flow Analysis Attack

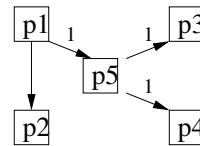
assuming that the nodes in the VoIP network may be malicious. We use a semi-honest (honest but curious) model for the malicious Skype peers. For instance, a malicious Skype peer may reveal the mapping between its in-flows and out-flows thereby reducing the anonymity of a VoIP flow. We discuss two other inference attacks on the AASIP protocol and sketch solutions to mitigate them.

#### 4.4.1 Flow Analysis Attacks

In this section, we describe three statistical inference based flow analysis attacks on VoIP networks. These attacks exploit the flow information and the shortest path nature of the voice paths to identify pairs of caller and receiver on the VoIP network. We assume that the adversary is aware the VoIP network topology. We represent the VoIP network topology as a weighted graph  $G = \langle V, E \rangle$ , where  $V$  is the set of nodes and  $E \subseteq V \times V$  is the set of undirected edges. The weight of an edge  $e = (p, q)$  (denoted by  $w(p, q)$ ) is the latency between the nodes  $p$  and  $q$ . We assume that the adversary knows  $nf(p \rightarrow q)$  the number



**Figure 64:** Tracing Voice Calls



**Figure 65:** Shortest Path Tracing

of voice flows between two nodes  $p$  and  $q$  on the VoIP network such that  $(p, q) \in E$ .

#### 4.4.1.1 Tracing Algorithm

Let us consider a caller  $src$ . We use a Boolean variable  $f(p) \in \{0, 1\}$  to denote whether the node  $p$  is reachable from  $src$  using the measured flows on the VoIP network. One can determine  $f(p)$  for all nodes  $p$  in  $O(|E|)$  time as follows. The base case of the recursion is  $f(src) = 1$ . For any node  $q$ , we set  $f(q)$  to one if there exists a node  $p$  such that  $(p, q) \in E \wedge f(p) = 1 \wedge nf(p \rightarrow q) > 0$ .

```

TRACE(Graph  $G = \langle V, E \rangle$ , Caller  $src$ )
(1)  for each vertex  $v \in V$ 
(2)     $f[v] = 0$ ;  $label[v] = \mathbf{false}$ 
(3)  end for
(4)   $f[src] = 1$ ;  $label[src] = \mathbf{true}$ 
(5)  while pick a labeled vertex  $v$ 
(6)     $label[v] = \mathbf{false}$ 
(7)    for each node  $u$  such that  $(u, v) \in E$ 
(8)      if ( $f[u] = 0$ )
(9)         $f[u] = 1$ ;  $label[u] = \mathbf{true}$ 
(10)     end if
(11)  end for
(12) end while

```

**Figure 66:** Tracing Algorithm

Let us consider a sample topology shown in Figure 64. For the sake of simplicity assume that each edge has unit latency. The label on the edges in Figure 64 indicates the number of voice flows. A trace starting from caller  $p_1$  will result in  $f(p_1) = f(p_2) = f(p_3) = f(p_4) = f(p_5) = 1$ . Filtering out the VoIP proxy nodes ( $p_5$ ) and the caller ( $p_1$ ), the clients  $p_2$ ,  $p_3$  and  $p_4$  could be potential destinations for a call emerging from  $p_1$ .

However, this tracing algorithm did not consider the shortest path nature of voice routes. Considering the shortest path nature of voice paths leads us to conclude that  $p_2$  is not a possible receiver for a call from  $p_1$ . If indeed  $p_2$  were the receiver then the voice flow would have taken the shorter route  $p_1 \rightarrow p_2$  (latency = 1), rather than the longer route  $p_1 \rightarrow p_5 \rightarrow p_2$  (latency = 2) as indicated by the flow information. Hence, we now have only two possible receivers, namely,  $p_3$  and  $p_4$ .

#### 4.4.1.2 Shortest Path Tracing Algorithm

Evidently a simple flow based tracing algorithm may be result in a large number of potential receivers for a voice call. In this section, we describe techniques to generate a directed sub-graph  $G' = \langle V', E' \rangle$  from  $G$  which encodes the shortest path nature of the voice paths. Given a graph  $G$  and a caller  $src$ , we construct a sub-graph  $G'$  that only contains voice paths that respect the shortest path property. Figure 67 uses a breadth first search on  $G$  to compute  $G'$  in  $O(|E|)$  time.

One can formally show that the directed graph  $G'$  satisfies the following properties: (i) If the voice traffic from  $src$  were to traverse an edge  $e \notin E'$ , then it violates the shortest path property. (ii) All voice paths that respect the shortest path property are included in  $G'$ . (iii) The graph  $G'$  is acyclic; additionally, if there is no node  $v$  that two alternate shortest path from  $src$ , the graph  $G'$  is a spanning tree.

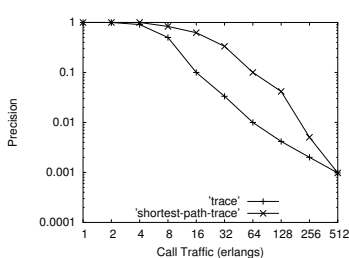
```

SHORTEST PATH TRACING(Graph  $G=\langle V, E \rangle$ , Caller
 $src$ )
(1)  for each vertex  $v \in V$ 
(2)     $dist[v] = \infty; label[v] = \text{false}; prev[v] = \text{null}$ 
(3)  end for
(4)   $dist[src] = 0; label[src] = \text{true}$ 
(5)  while pick a labeled vertex  $v$  with minimum
 $dist[v]$ 
(6)     $label[v] = \text{false}$ 
(7)    for each node  $u$  such that  $(u, v) \in E$ 
(8)      if  $(dist[u] < dist[v] + w(u, v))$ 
(9)         $dist[u] = dist[v] + w(u, v)$ 
(10)        $prev[u] = \{v\}; label[u] = \text{true}$ 
(11)      end if
(12)      if  $(dist[u] = dist[v] + w(u, v))$ 
(13)         $prev[u] = prev[u] \cup \{v\}$ 
(14)      end if
(15)    end for
(16)  end while
(17)  $G' = \langle V', E' \rangle: V' = V, E' = (u \rightarrow v) \forall u \in$ 
 $prev[v], \forall v \in V$ 

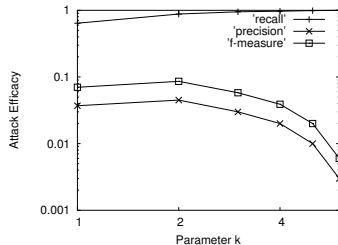
```

**Figure 67:** Shortest Path Tracing Algorithm

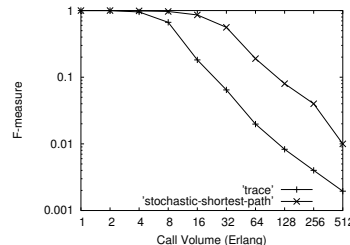
Figure 65 illustrates the result of applying algorithm 67 on Figure 64. Indeed if one uses



**Figure 68:** Shortest Path Tracing Vs Naive Tracing



**Figure 69:** Precision and Recall with 128 Erlang Call Volume



**Figure 70:**  $F$ -measure with  $\kappa = 2$

the trace algorithm (Figure 66) on graph  $G'$ , we get  $f(p_2) = 0$ ,  $f(p_3) = f(p_4) = 1$ . Figure 68 compares the effectiveness of shortest path tracing algorithm with the naive tracing algorithm on a 1024 node VoIP network. On the x-axis we plot the call traffic measured in Erlang (calls per unit time) [46]. We use two parameters to quantify the efficacy of an attack: *precision*, *recall* and *F-measure* defined below. An attack is very effective if its precision and recall is high. We use  $S$  to denote the set of nodes such that for every  $p \in S$ ,  $f[p] = 1$ . Recall denotes the probability of identifying the true caller  $dst$  and precision is inversely related to the size of candidate receiver set. One trivially improve recall by including all VoIP clients in the set  $S$ ; however, this would adversely affect the precision metric.  $F$ -measure is harmonic mean of recall and precision scores.

$$\begin{aligned}
 recall &= Pr(f[dst] = 1) \\
 precision &= \begin{cases} \frac{1}{|S|} & \text{if } dst \in S \\ 0 & \text{otherwise} \end{cases} \\
 F\text{-measure} &= \frac{2 * recall * precision}{recall + precision}
 \end{aligned}$$

In a deterministic network setting, the receiver  $dst$  is guaranteed to be marked with  $f[dst] = 1$ , that is,  $recall = 1$  for both the naive tracing algorithm and the shortest path tracing algorithm. However, the shortest path tracing algorithm is significantly more precise (see Figure 68) than the naive tracing algorithm.

#### 4.4.1.3 Stochastic Shortest Path Tracing Algorithm

In a realistic setting with uncertainties in network latencies the shortest path tracing algorithm may not identify the receiver. We handle such uncertainties in network link latencies, by using the top- $\kappa$  shortest path algorithm to construct  $G^\kappa$  from  $G$ . An edge  $e$  is in  $G^\kappa$  if it is permissible to appear on the top- $\kappa$  shortest paths originating from  $src$  in graph  $G$ . One can accommodate this change by simply maintaining top- $\kappa$  distance measurements  $dist^1[v]$ ,  $dist^2[v]$ ,  $\dots$   $dist^\kappa[v]$  instead of only the shortest (top-1) distance measurement. Evidently, as  $\kappa$  increases, the tracing algorithm can accommodate higher uncertainty in network latencies, thereby improving *recall*. On the other hand, as  $\kappa$  increases, the number of candidate receivers become larger, thereby decreasing the *precision* metric.

One can formally show that the directed graph  $G^\kappa$  satisfies the following properties: (i) If the voice traffic from  $src$  were to traverse an edge  $e \notin E'$ , then it violates the top- $\kappa$  shortest path property. (ii) All voice paths that respect the top- $\kappa$  shortest path property are included in  $G^\kappa$ . (iii) Unlike  $G^1$ , the graph  $G^\kappa$  (for  $\kappa \geq 2$ ) may contain directed cycles.

We used the GT-ITM [125] topology generator to construct a 1024 node VoIP network topology. Figure 69 shows the precision, recall and  $F$ -measure of the stochastic shortest path tracing algorithm with 128 Erlang call volume and varying  $\kappa$ . These experiment lead us to conclude that  $\kappa = 2$  yields a concise (high *precision*) and yet precise (high *recall*) list of potential receivers. Figure 70 compares the  $F$ -measure for the stochastic shortest path tracing algorithm and the naive tracing algorithm using  $\kappa = 2$  and varying call volume.

#### 4.4.1.4 Flow Analysis Algorithm

We have so far used a Boolean variable  $f(p)$  to denote whether a VoIP client  $p$  can be a potential receiver for a VoIP call from  $src$ . We use the flow measurements to construct a probability distribution over the set of possible measurements. Let  $G^\kappa$  be a sub-graph of  $G$  obtained using the top- $\kappa$  shortest path tracing algorithm with caller  $src$ . Let  $nf(p \rightarrow q)$  denotes the number of flows on the edge  $p \rightarrow q$ . Let  $in(p)$  denote the total number of flows into peer  $p$ . Note that both  $nf(p \rightarrow q)$  and  $in(p)$  are observable by an external adversary. Assuming a peer  $p$  in the VoIP network performs perfect mixing, the probability that some

incoming flow is forwarded on the edge  $p \rightarrow q$  as observed by an external adversary is  $\frac{nf(p \rightarrow q)}{in(p)}$ . Let  $f(p)$  denote the probability that a VoIP flow originating at  $src$  flows through peer  $p$ . The function  $f$  is recursively defined on the directed edges in  $G^\kappa = \langle V^\kappa, E^\kappa \rangle$  as follows:

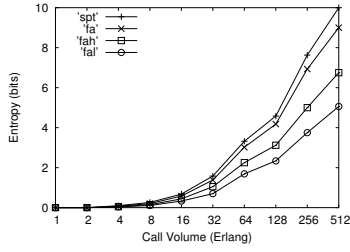
$$f(q) = \sum_{p \rightarrow q \in E^\kappa} f(p) * \frac{nf(p \rightarrow q)}{in(p)} \quad (13)$$

with the base case  $f(src) = 1$  and  $in(src) = 1$ . Now, every VoIP client  $p$  ( $p \neq src$ ) is a possible destination for the VoIP flow originating from  $src$  if  $f(p) > 0$ . Consistent with other work in this research area [70] one can use entropy  $E = -\sum_p f(p) * \log_2 f(p)$  as a metric for anonymity. In addition, one might use the top- $m$  probability metric, namely, the probability that the receiver  $dst$  appears in the top- $m$  entries when  $f(p)$  is sorted in descending order.

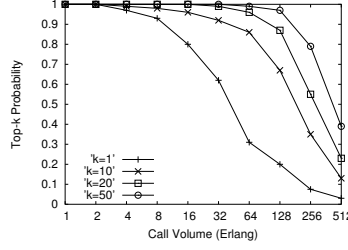
Computing the probabilities  $f(p)$  for  $G^1$  (top-1 shortest paths) is very efficient. Observe that since  $G^1$  is a directed acyclic graph, it can be sorted topologically. Let  $p_1 = src, p_2, \dots, p_N$  be a topological ordering of the nodes in  $G^1$  such that  $f(p_i)$  depends on  $f(p_j)$  only if  $j < i$ . Hence, one can efficiently evaluate the probabilities by following the topological order, namely, compute  $f(p_1), f(p_2), \dots, f(p_N)$  in that order. However,  $G^\kappa$  (for  $\kappa \geq 2$ ) contains cycles and thus cannot be topologically sorted. In this case, we represent the set of equations in 13 as  $\pi = \pi M$ , where  $\pi$  is a  $1 \times N$  row vector and  $M$  is a  $N \times N$  matrix, where  $\pi_i = f(p_i)$  and  $M_{ij} = \frac{nf(p_i \rightarrow p_j)}{in(p_i)}$  if there exists a directed edge  $p_i \rightarrow p_j$  in  $G^\kappa$ ; and  $M_{ij} = 0$  otherwise. Hence, the solution  $\pi$  is the stationary distribution of a Markov chain whose transition probability matrix is given by  $M$ . We compute  $\pi$  iteratively:  $\pi^{t+1} = \pi^t M$  starting with  $\pi_i^0 = 1$  if  $p_i = src$ ;  $\pi_i^0 = 0$  otherwise. Assuming  $M$  is irreducible,  $\pi$  converges to a steady state solution for the equation  $\pi = \pi M$  in about  $O(N \log N)$  iterations.

#### 4.4.1.5 Distance and Hop Count Prior

In this section, we enhance the efficacy of flow analysis algorithm using hop count and distance prior. Let us suppose that  $g_{hop}(x)$  denotes a probability density function over the number of overlay network hops (along the shortest path) between a randomly chosen pair of caller  $src$  and receiver  $dst$ . Let  $g_{lat}(x)$  denote a probability distribution function over



**Figure 71:** Entropy with  $\kappa = 2$



**Figure 72:** Top- $m$  Probability with  $\kappa = 2$  and Latency Prior

$\kappa$	# Iterations	Time (ms)
1	-	0.03
2	7	31
3	11	49.7
4	19	84.2

**Figure 73:** Computation Cost

the one-way latency (along the shortest path) between a randomly chosen pair of caller  $src$  and receiver  $dst$ . One can use a priori information about distance and hop count to compute  $f(p)$ , namely, the probability that a VoIP flow originating from  $src$  flows through peer  $p$ . Using the hop count prior, the probability that a node  $p$  forwards an incoming flow on the edge  $p \rightarrow q$  is  $\frac{nf(p \rightarrow q)}{in(p)} * Pr(hop \geq hopdist(src, p) + 1 \mid hop \geq hopdist(src, p))$ , where  $hopdist(src, p)$  denotes the number of hops along the shortest path between  $src$  and  $p$  on graph  $G'$ .  $Pr(hop \geq hopdist(src, p) + 1 \mid hop \geq hopdist(src, p)) = \frac{Pr(hop \geq hopdist(src, p) + 1)}{Pr(hop \geq hopdist(src, p))}$  denotes the probability that the receiver  $dst$  could be one more hop away given that  $dst$  is at least  $hopdist(src, p)$  hops away from  $src$ .

Using the distance prior, the probability that a node  $p$  forwards an incoming flow on the edge  $p \rightarrow q$  is  $\frac{nf(p \rightarrow q)}{in(p)} * Pr(lat \geq latdist(src, p) + w(p, q) \mid lat \geq latdist(src, p))$ , where  $latdist(src, p)$  denotes the latency of the shortest path between  $src$  and  $p$  on graph  $G'$  and  $w(p, q)$  denotes the one-way latency between nodes  $p$  and  $q$ .  $Pr(lat \geq latdist(src, p) + w(p, q) \mid lat \geq latdist(src, p)) = \frac{Pr(lat \geq latdist(src, p) + w(p, q))}{Pr(lat \geq latdist(src, p))}$  denotes the probability that the receiver  $dst$  could be  $w(p, q)$  time units further away given that  $dst$  is at least  $latdist(src, p)$  time units away from  $src$ . As in the flow analysis algorithm, the function  $f$  is defined on the directed edges in graph  $G^\kappa = \langle V^\kappa, E^\kappa \rangle$  as follows:

$$f(q) = \sum_{p \rightarrow q \in E^\kappa} f(p) * \frac{nf(p \rightarrow q)}{in(p)} * \frac{Pr(hop \geq hopdist(src, p) + 1)}{Pr(hop \geq hopdist(src, p))}$$

$$f(q) = \sum_{p \rightarrow q \in E^\kappa} f(p) * \frac{nf(p \rightarrow q)}{in(p)} * \frac{Pr(lat \geq latdist(src, p) + w(p, q))}{Pr(lat \geq latdist(src, p))}$$

We compute the probability that number of hops is greater than  $x$  by  $Pr(hop \geq x) =$

$\sum_{y=x}^{\infty} g_{hop}(y)$  and the probability that the one-way latency is greater than  $x$  by  $Pr(lat \geq x) = \int_x^{\infty} g_{lat}(y)dy$ . We estimate the probability distributions  $g_{hop}$  and  $g_{lat}$  by measuring the distances between two randomly chosen pair of  $src$  and  $dst$  nodes on the VoIP network. Similar to the flow analysis attack, we can compute  $f(p)$  using a topological sort on acyclic directed graph  $G^1$  and the Markov chain model for  $G^\kappa$  ( $\kappa \geq 2$ ).

Figure 71 shows the entropy of an attack versus call volume using  $\kappa = 2$ . Note that smaller the entropy better is the efficacy of the attack in identifying a precise and concise set of potential receivers. We observe that the flow analysis algorithm with latency prior offers the best results. This is primarily because the Skype lookup protocol always constructs voice paths that have minimum one-way latency. The latency prior reflects the latency based shortest path nature of the Skype lookup protocol and thus performs best.

Figure 72 shows the top- $m$  probability, namely, the probability that the true receiver  $dst$  appears in the top- $m$  entries when  $f(p)$  is sorted in descending order using the flow analysis algorithm with latency prior and  $\kappa = 2$ . With a call volume of 64 Erlangs, there is 86% chance that the true receiver  $dst$  appears in the top-10 entries. This clearly demonstrates the efficacy of our attacks in identifying the receiver under low to moderate call volume. Under very high call volume the attack (512 Erlangs) the top-10 probability drops to 0.17.

Figure 73 shows the computation cost (in time units) incurred in computing the probabilities  $f(p)$  for all potential receivers  $p$ . In practice, we observed that the number of iterations is much smaller than the theoretical bound  $O(N \log N)$ . We attribute this to the sparse nature of matrix  $M$ . The overall running time is of the order of few tens of milliseconds making the attack feasible in real-time. In summary, the flow analysis algorithm allows an external observer to deduce a reasonably concise and yet precise set of potential receivers for a voice call. One could additionally incorporate other prior knowledge such as the geographical location of the receiver, duration of the call, etc.

#### 4.4.1.6 Compromised Proxies

In addition, to passive observation based attacks, the adversary could compromise some of the nodes in the VoIP proxy. A compromised proxy node reveals its mixing information to

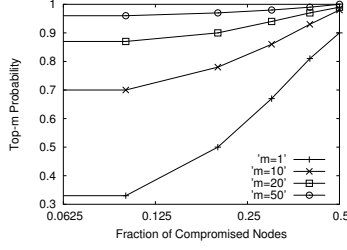
an adversary. The adversary can use this information from one or more compromised to enhance the efficacy of its attack as follows. A malicious proxy node may reveal the routing information that relates its incoming VoIP flows to out going VoIP flows. We use  $f(p \rightarrow q)$  to denote the probability that a VoIP call from  $src$  traverses the edge  $p \rightarrow q$ . For a non-malicious node  $p$ ,  $f(p \rightarrow q) = f(p) * \frac{nf(p \rightarrow q)}{in(p)}$ , where  $f(p)$  is the probability that the VoIP call from  $src$  traverses node  $p$ . For a malicious node  $p$ ,  $f(p \rightarrow q) = \sum_{r \rightarrow p} f(r \rightarrow p) * \frac{nf(r \rightarrow p \rightarrow q)}{nf(r \rightarrow p)}$ , where  $nf(r \rightarrow p \rightarrow q)$  denotes the number of voice flows that were routed from  $r$  to  $q$  by the malicious node  $p$ . Observe that a good proxy  $p$ ,  $nf(r \rightarrow p \rightarrow q)$  is neither revealed by  $p$  nor observable (using traffic analysis) to an external adversary. Hence, the new flow analysis equations are given as follows:

$$\begin{aligned}
 f(p \rightarrow q) &= \begin{cases} f(p) * \frac{nf(p \rightarrow q)}{in(p)} & \text{honest } p \\ \sum_{r \rightarrow p} f(r \rightarrow p) * \frac{nf(r \rightarrow p \rightarrow q)}{nf(r \rightarrow p)} & \text{compromised } p \end{cases} \\
 f(q) &= \sum_{p \rightarrow q} f(p \rightarrow q)
 \end{aligned} \tag{14}$$

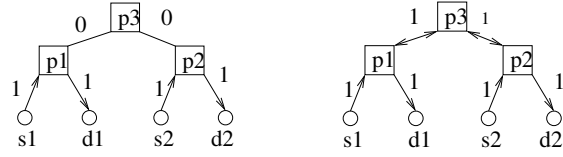
One should note that these equations are identical to Equation 13 when there are no compromised proxies in the VoIP network. We can compute  $f(p)$  (for all nodes  $p$ ) using a topological sort on acyclic directed graph  $G^1$  and the Markov chain model for  $G^\kappa$  ( $\kappa \geq 2$ ). Figure 74 shows the effectiveness of the attack as we vary the fraction of nodes that is compromised by the adversary. We use a call volume of 128 Erlangs and  $\kappa = 2$  in this experiment. For instance, when 20% of the nodes are compromised the top-1 probability improves from 0.23 to 0.50.

#### 4.4.2 VoIP Privacy using $k$ -Anonymity

In this section, we focus on developing a distributed and decentralized algorithm to better protect the identity of a receiver from a flow analysis attack. First, we observe that the efficacy of our attacks drops significantly under high call volume. This is primarily because larger call volume facilitates better mixing of VoIP flows and thus better protects the identity of a receiver from flow analysis attacks. In this chapter, we propose solutions to construct VoIP paths that allow a large number of VoIP flows to be mixed together. Our



**Figure 74:** Compromised Proxies



**Figure 75:** 1-anonymity

**Figure 76:** 2-anonymity

proposal is based on the following observation. In a VoIP network, one-way latency up to 250ms is not even perceived by the clients, while latencies in the range 250-400ms is acceptable for voice conversations [95]. Hence, one could construct sub-optimal voice paths whose one-way path latency is smaller than  $maxLat = 250ms$  without affecting the quality of voice conversations.

Figures 75 illustrate a simple scenario with two VoIP flows between clients  $(s_1, d_1)$  and  $(s_2, d_2)$ . Figure 76 shows how one can modify VoIP routes to achieve 2-anonymity. In the figures, each edge on the VoIP network is marked with the number of flows it carries. We say that a voice call is  $k$ -anonymous, if the voice flow is mixed with at least  $k - 1$  other flows. One can show that given a voice route  $p_1 = src, p_2, \dots, p_m = dst$ , the number of flows mixed with the VoIP call from  $src$  to  $dst$  is bounded by Equation 15.

$$max_i \{in(p_i)\} \leq k \leq 1 + \sum_{i=2}^m in(p_i) - nf(p_{i-1} \rightarrow p_i) \quad (15)$$

Observe that routing all the VoIP calls through one peer would provide very high anonymity (assuming the peer is honest). However, peers need to maintain per-flow state to handle routing information, handle peer failures, joins and leaves, and support several advanced calling features [2]. In the rest of this section, we modify the Skype lookup protocol to set up anonymity-latency-capacity-aware routes assuming that all the proxies are honest and uncompromised. We study potential attacks on our protocol by malicious peers and propose defenses to mitigate them in Section 4.4.2.3.

#### 4.4.2.1 AASIP: Anonymity-Aware Session Initiation Protocol

In this section, we present our AASIP protocol. We accept the anonymity parameter  $k$  as an input for the lookup protocol, on a per-client per-call basis. The AASIP protocol modifies the basic Skype search protocol such that it simultaneously satisfies three conditions: (i) we have at least one peer  $p \in route(src, dst)$  such that  $in(p) \geq k$  ( $k$ -anonymity), (ii) the end-to-end one-way latency on the route from  $src$  to  $dst$  is smaller than  $maxLat$  (typically set to 250ms), and (iii) the total call volume on every peer  $p \in route(src, dst)$  is smaller than its capacity  $maxFlow(p)$ . We assume that all search traffic between peers  $p$  and  $q$  are encrypted with a symmetric key shared between  $p$  and  $q$ . The search traffic being significantly smaller than the voice traffic can be easily hidden from an external observer using a uniform cover traffic [37].

A naive protocol operates in two steps. The first step uses a Skype like lookup protocol to locate a peer  $p$  that is closest to  $src$  such that  $in(p) \geq k$ , that is, peer  $p$  mixes at least  $k$  voice flows. The second step uses the peer  $p$  to search for  $dst.sipurl$ . The route from  $src$  to  $p$  concatenated with that from  $p$  to  $dst$  satisfies  $k$ -anonymity. However, such a protocol may not essentially result in a low latency route from  $src$  to  $dst$ . In the rest of this section, we present the AASIP protocol that integrates anonymity and latency awareness. Similar to the Skype lookup protocol, the AASIP protocol operates in four phases: `initSearch`, `processSearch`, `processResult`, and `finSearch`.

**initSearch.** A VoIP client  $src$  initiates a search for a receiver  $dst$  (identified by its SIP URL) and an anonymity parameter  $k$  by broadcasting  $search(searchId, sipurl = dst.sipurl, k, anonVec=[1])$  to all peers  $p \in ngh(src)$ . The search identifier  $searchId$  is a long randomly chosen unique identifier.

**processSearch.** Let a peer  $p$  receive a request  $req = search(searchId, sipurl, k, anonVec)$  from its neighbor  $q$ . If the call volume at peer  $p$  exceeds  $maxFlow(p)$ , then peer  $p$  drops the request. If peer  $p$  has seen  $searchId$  in the recent past then it extracts the cached request entries  $reqC = \langle searchId, sipurl, prev, k, anonVec \rangle$ . If for any cached request entry  $c \in reqC$ ,  $c.anonVec$  satisfies  $k$ -anonymity or if  $c.anonVec \geq req.anonVec$  then the

request is dropped. If peer  $p$  has seen  $searchId$  in the recent past then it extracts the cached result entries  $resC = \langle searchId, next, k, anonVec \rangle$ . If for any cached result entry  $c \in resC$ ,  $c.anonVec$  satisfies  $k$ -anonymity or if  $c.anonVec \geq req.anonVec$  then the request is dropped. If the request is not dropped, peer  $p$  adds  $\langle searchId, sipurl, q, k, anonVec' \rangle$  to its request cache  $reqC$ , where  $anonVec' = req.anonVec.append(in(p))$ .

If peer  $p$  were to not have seen  $searchId$  in the past, peer  $p$  checks if  $sipurl$  is the URL of a VoIP client connected to  $p$ . If so, peer  $p$  returns its IP address  $result(searchId, p, anonVec')$  to peer  $q$  and adds the result to its cache  $resC$ . If not, peer  $p$  broadcasts  $search(searchId, sipurl, k, anonVec')$  to all peers  $p' \in ngh(p)$ , including the neighbor  $q$  from whom it received the request. As illustrated in Figures 75 and 76 it might be beneficial to set up loops in VoIP routes for achieving higher anonymity. However, the request dropping constraints described above ensures that no loop is traversed more than once. Finally, peer  $p$  adds  $\langle searchId, sipurl, q, k, anonVec' \rangle$  to its request cache  $reqC$ .

**processResult.** Let a peer  $p$  receive a result  $res = result(searchId, q, anonVec)$  from peer  $q$ . If peer  $p$  has seen  $searchId$  in the recent past then it extracts the cached result entries  $resC = \langle searchId, next, k, anonVec \rangle$ . If for any cached result entry  $c \in resC$ ,  $c.anonVec$  satisfies  $k$ -anonymity or if  $c.anonVec \geq res.anonVec$  then the result  $res$  is dropped. If peer  $p$  has seen  $searchId$  in the recent past then it extracts the cached request entries  $reqC = \langle searchId, sipurl, prev, k, anonVec \rangle$ . If for any cached request entry  $c \in reqC$ ,  $c.anonVec$  satisfies  $k$ -anonymity or if  $c.anonVec \geq res.anonVec$  then the result is dropped. If the result  $res$  has not been dropped, peer  $p$  adds a routing entry  $\langle sipurl, prev, q \rangle$  and forwards  $result(searchId, p, anonVec)$  to peer  $prev$ . Unlike the Skype lookup protocol, the routing entry is a three tuple  $\langle sipurl, prev, next \rangle$ , which indicates that voice packets from peer  $prev$  destined to  $sipurl$  must be forwarded to  $next$ . The three tuple routing table allows us to handle loops in routing table entries wherein the next hop depends not only on the destination  $sipurl$  but also on the previous hop. For example, from Figure 76 peer  $p_3$  may have the following two routing entries:  $\langle d_1, p_1, p_2 \rangle$  and  $\langle d_1, p_2, p_1 \rangle$ .

**finSearch.** When the peer  $ngh(src)$  receives  $res = result(searchId, q, anonVec)$  from

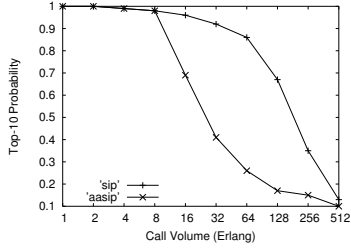
peer  $q$ , it extracts the cached result entries  $resC = \langle searchId, next, k, anonVec \rangle$ . If for any cached result entry  $c \in resC$ ,  $c.anonVec$  satisfies  $k$ -anonymity or if  $c.anonVec \geq res.anonVec$  then the result  $res$  is dropped. If not  $ngh(src)$  adds a routing entry  $\langle dst, *, q \rangle$  and adds  $res$  to its result cache  $resC$ .

The route set up protocol ensures that when a voice session is initialized, the route from  $src$  to  $dst$  satisfies  $k$ -anonymity. We ensure that  $k$ -anonymity during a voice session as follows. We piggy back the anonymity vector  $anonVec$  along with the voice packets as they are tunneled through the VoIP network along the designated route. If the VoIP route were to violate the  $k$ -anonymity constraint, the caller could be warned of the lower level of anonymity guaranteed by the route. We allow the VoIP client to set a lower watermark  $k' \leq k$ , such that if the anonymity level drops below  $k'$  the session is terminated. Now, the caller  $src$  may initiate a new route set up protocol with the goal of achieving the required anonymity level. In our experimental section we study the probability that a VoIP session is terminated due to a drop in the anonymity level. In addition, one can construct VoIP routes that include multiple nodes that provide  $k$ -anonymity; hopefully, at least one of these peers would offer  $k$ -anonymity for the entire duration of the VoIP session. We examine this solution in greater detail in Section 4.4.2.3.

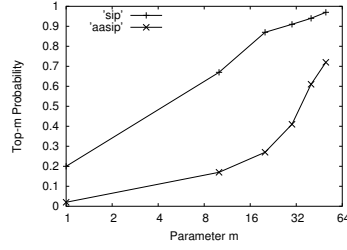
The AASIP protocol sets up the fastest possible route from  $src$  to  $dst$  that meets  $k$ -anonymity (if there exists such a route). If no such route exists, the AASIP protocol set up identifies a route with highest possible anonymity  $k' < k$ ; in addition, the AASIP protocol sets up the fastest route that achieves  $k'$ -anonymous route from  $src$  to  $dst$ . Indeed, if one sets  $k = \infty$ , then the AASIP protocol identifies the most anonymous route whose one-way latency is smaller than  $maxLat = 250ms$ . Our experiments show that the AASIP protocol is highly efficient in terms of its messaging cost and route set up latency and protects a VoIP call from flow analysis attacks.

#### 4.4.2.2 Flow Analysis Attacks on the AASIP Protocol

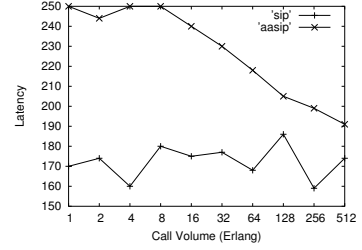
A flow analysis attack on the AASIP protocol operates in three phases. We assume that the adversary knows the parameter  $k$  used by a caller  $src$ . First, the adversary identifies



**Figure 77:** AASIP Vs SIP: Top-10 Probability



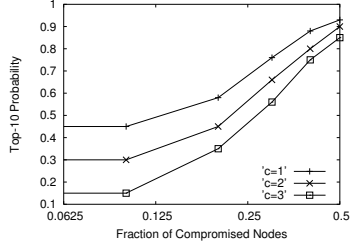
**Figure 78:** AASIP Vs SIP: 128 Erlangs



**Figure 79:** AASIP Vs SIP: Path Latency

all nodes  $p$  such that  $in(p) \geq k$ . Let  $G_{src}^{\kappa}$  be a sub-graph of  $G$  obtained using the top- $\kappa$  shortest path tracing algorithm with caller  $src$ . We use  $G_{src}^{\kappa}$  and the flow measurements to compute  $f_{src}(p)$  for all  $p$  such that  $in(p) \geq k$  starting with  $f_{src}(src) = 1$ . Second, for every such node  $p$  with  $in(p) \geq k$ , we construct  $G_p^{\kappa}$  as a sub-graph of  $G$  obtained using the top- $\kappa$  shortest path tracing algorithm with caller  $p$ . We compute  $f_p(r)$  for every candidate receiver  $r$  starting with  $f_p(p) = f_{src}(p)$ , where  $f_{src}(p)$  is obtained from the first step. Third, we compute  $f(r)$  for every receiver (VoIP clients) as a simple average of  $f_p(r)$  over all such nodes  $p$ . Similar to other flow analysis attacks, one could sort the receivers in descending order of  $f(r)$  and use a top- $m$  probability metric to study the efficacy of the attack.

Figures 77 and 78 compares the effectiveness of the AASIP protocol with the basic SIP protocol in mitigating flow analysis attacks. We set the parameter  $k = \infty$  so that the AASIP protocol identifies the most anonymous route from caller  $src$  to receiver  $dst$  that satisfies the latency constraint. Figure 77 shows the top-10 probability using both the AASIP and SIP protocol for varying call volumes. Observe that at low and moderate call volumes the AASIP protocol offers significantly improved protection against flow analysis attacks. Figure 78 shows the top- $m$  probability for both the AASIP and the SIP protocol for varying  $m$  and a call volume of 128 Erlangs. We observe that the AASIP protocol consistently outperforms the SIP protocol for all values of  $m$ . Figure 79 compares the average one-way path latency for both the AASIP and the SIP protocol for varying call volumes. At low call volumes the AASIP protocol may construct significantly longer routes than the SIP protocol. Nonetheless, the AASIP protocol ensures that the latency is below 250ms and thus preserves the quality of the voice session.



**Figure 80:** Compromised Proxies

$k$	Time (seconds)
1	0.03
10	2.9
20	4.0
50	5.9

**Figure 81:** Computation Cost:  $k$ -Anonymity

$c$	Time (seconds)
1	2.9
2	4.4
3	11.6
4	43.5

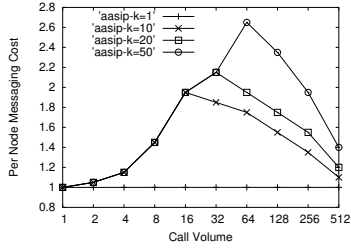
**Figure 82:** Computation Cost: Tolerating Malicious Proxies

#### 4.4.2.3 Tolerating Compromised Proxies

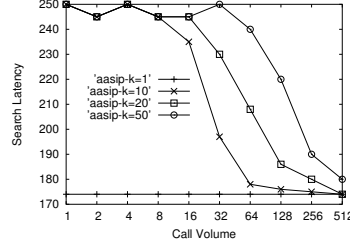
In this section, we study the effect of compromised proxies on the AASIP protocol. Similar to Section 4.4.1, we assume that a compromised peer will execute the AASIP protocol honestly. However, the malicious peers may record observed information during the voice session and collude with the external observer. In particular, a malicious Skype peer may reveal the mapping between its in-flows and out-flows thereby decreasing the anonymity of a VoIP flow.

We present a simple extension to the AASIP protocol to tolerate malicious peers and yet preserve  $k$ -anonymity. We allow the caller  $src$  to specify a personalized security parameter  $c$  for every VoIP call indicating that the route from  $src$  to  $dst$  should tolerate up to  $c$  compromised nodes while preserving  $k$ -anonymity. The key idea is to construct a route from  $src$  to  $dst$  with at least  $c + 1$  peers  $p_1, p_2, \dots, p_{c+1}$  such that  $in(p_i) \geq k$  for all  $1 \leq i \leq c + 1$ . One can introduce this constraint into the AASIP protocol by replacing the constraint  $anonVec \geq k$  with  $anonVec \geq (k, c)$ . Note that  $anonVec \geq k$  denotes that there exists an index  $j$  such that  $anonVec[j] \geq k$ . We use  $anonVec \geq (k, c)$  to denote that there exists  $c + 1$  indices  $j_1, j_2, \dots, j_{c+1}$  such that  $anonVec[j_i] \geq k$  for all  $1 \leq i \leq c + 1$ .

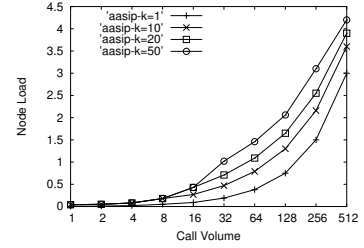
One can extend the flow analysis attack on the  $k$ -anonymous SIP protocol to a  $(k, c)$ -anonymous SIP protocol. We assume that the adversary knows the parameters  $k$  and  $c$  used by a caller  $src$ . The adversary identifies a set of nodes  $S$  such that for all nodes  $p \in S$ ,  $in(p) \geq k$ . For every permutation of  $c$  nodes from the set  $S$ , the adversary computes  $f_{src, p_1, p_2, \dots, p_c}(r)$  for a candidate receiver  $r$ . We recursively compute  $f_{p_1, p_2, \dots, p_i}(p)$  using flow analysis algorithms on  $G_{p_i}^k$  and starting probability  $f_{start}(p_i) = f_{p_1, p_2, \dots, p_{i-1}}(p_i)$ . The



**Figure 83:** Messaging Cost



**Figure 84:** Search Latency



**Figure 85:** Node Load

recursion terminates at the base case  $f_{start}(src) = 1$ . Finally, we compute  $f(r)$  as the average of  $f_{src,p_1,p_2,\dots,p_c}(r)$  over all ordered selections of  $c$  nodes from the set  $S$ . This attack reduces a simple flow analysis attack on the AASIP protocol when  $c = 1$ .

Figure 80 shows the effectiveness of the attack as we vary the fraction of nodes that is compromised by the adversary and the parameter  $c$ . We use a call volume of 128 Erlangs,  $k = 10$  and  $\kappa = 2$  in this experiment. We observe that as  $c$  increases the effectiveness of the attack decreases significantly. Nonetheless, when an overwhelming large number of nodes are malicious the AASIP protocol becomes vulnerable to flow analysis attacks. Under a realistic setting, only a small number of nodes ( $< 10\%$ ) are likely to be malicious, the AASIP protocol offers very good protection against flow analysis attacks.

### 4.4.3 Experimental Evaluation

#### 4.4.3.1 Implementation Sketch

In this section, we describe a brief implementation of our algorithms using Phex [1]: an open source Java based implementation of peer-to-peer broadcast search protocol. A VoIP protocol like Skype operates on top of the peer-to-peer infrastructure. We have implemented our algorithms as pluggable modules that can be weaved into the Phex client code using AspectJ [33]. Our implementation is completely transparent to the VoIP protocol that operates on top of the peer-to-peer infrastructure. Also, our implementation does not require any changes to topology construction and maintenance algorithms (as nodes join, leave, fail or recover) and the underlying TCP/IP or UDP based communication libraries.

Below we sketch our implementation of the AASIP protocol. As described in Section 4.2,

a broadcast search protocol has four parts: `initSearch`, `processSearch`, `processResult` and `finSearch`. These four operations are implemented as event handlers in Phex. When a Phex client receives a messages, it determines the type of the message (search request, search result, etc) and triggers the appropriate event handler. The AASIP protocol substitutes all four event handlers (see Section 4.4.2.1). In addition, we also modify the search request payload and the search result payload to include the parameters  $k$ ,  $c$  and  $anonVec$ .

#### 4.4.3.2 Performance and Scalability

In this section, we compare the messaging cost and the search latency of the AASIP and the SIP protocol. Figure 83 shows the messaging cost per node as we vary the call volume and the anonymity parameter  $k$  (using  $c = 1$ ). These experiments show that the AASIP protocol incurs about 1-3 times the messaging cost of the SIP protocol. However, the search request and the results are typically of the order of 300 Bytes. Hence, the communication cost at a node for handling 10 messages (3 KB) is equivalent to a voice session of one second (24 Kbps). Even though the AASIP protocol incurs higher communication cost than the SIP protocols, its effect on the overall bandwidth consumption is negligible.

Figure 84 shows the latency of a search operation as we vary the call volume and the anonymity parameter  $k$  (using  $c = 1$ ). This experiment shows that the AASIP protocols incurs about 30-40% higher search latency than the SIP protocol. One should note that the search latency only affects the initial waiting time before the caller and the receiver are connected. Once a path is established the AASIP protocol ensures good quality voice conversations by limiting the path latency to 250ms.

Figure 85 shows the average number of VoIP calls handled by one VoIP proxy in the VoIP network (using  $c = 1$ ). The AASIP protocol incurs a higher load primarily because the traffic is not routed through the optimal route. Hence, a voice route in the AASIP protocol may include more network hops than the SIP protocol. This increases the average number of proxies that route one voice call, and thus increase the average load on a proxy. The percentile increase in average node load for higher call volumes is small. Hence, when the call volume is high (VoIP network is heavily loaded), the AASIP protocol imposes small

overheads (20-40%) than the SIP protocol. On the other hand, when the call volume is low (VoIP network is lightly loaded), the AASIP protocol incurs 4-6 times the average node load when compared to the SIP protocol. However, this 4-6x increase in node load is incurred when the VoIP network is itself lightly loaded; hence, the AASIP protocol does not harm the performance and scalability of the VoIP network.

## 4.5 *Related Work*

Several chapters have addressed the problem of tracing encrypted traffic using timing analysis [112, 122, 126, 111, 14, 29, 110]. All these chapters use inter-packet timing characteristics for tracing traffic. The timing characteristics may be obtained either by passively observing the traffic or by actively embedding a timing signature in the traffic. Our approach entirely differs from the above approaches since it uses network topology based timing measurements to device caller identification attacks.

A large number of low latency anonymizing networks have been proposed. Onion routing [40] and its second generation Tor [28] aim at providing anonymous transport of TCP flows over the Internet. ISDN mixes [71] proposes solutions to anonymize phone calls over traditional PSTN. While anonymizing networks protect the identity of the sender and receiver using the concept of a mix [19, 91]. However, they assume that the sender *src* and the receiver *dst* is known a priori. The above protocols do not take the search phase (discovering the receiver *dst*) into account.

Tarzan [37] presents an anonymizing network layer using the peer-to-peer model. It uses a gossip based approach to disseminate the identity of nodes on the peer-to-peer network. The gossip protocol propagates  $\langle sipurl, IPAddr \rangle$  information to all the nodes in the network. At the end of the gossip protocol, the caller *src* supposedly knows the receiver *dst*, thus obviating the need for a search protocol. This approach requires the caller and receiver to be connected to the network sufficiently long before they can discover each other (and make VoIP calls). Most peer-to-peer VoIP applications (like Skype [2]) use KaZaa like broadcast search algorithm to discover the receiver *dst* and set up a low latency voice path on the network.

Traditionally, multicast and broadcast protocols have been used to protect receiver anonymity [72]. This approach sends a unicast message to a single destination by multicasting the message to a group containing the destination. The intended destination recognizes the message is intended for itself because it expects this message (Hordes [90]) or because the sender addresses the message implicitly (as in Section 4.3.2.3, [72]). In contrast to these pieces of work, our chapter focuses on caller (source) anonymity as against receiver anonymity.

Perng et. al. [70] have shown that multicasting data on an anonymizing network can break some privacy guarantees. In particular, they show that the malicious nodes in the network can use the popularity of a multicast packet to infer some information about the data. Our attacks exploit the underlying multicast/broadcast nature of the search protocol to construct triangulation based timing attacks. Unlike using the popularity of the multicast data, we use network topology based timing information to infer the caller.

## ***4.6 Summary***

In this chapter we have addressed the problem of tracing the caller in encrypted peer-to-peer VoIP networks. We have developed two attacks: one on the SIP protocol layer and the second on the voice session layer. These attacks can be implemented by passively observing the search traffic on the VoIP network. We have demonstrated the effectiveness of these attacks using the popular Skype VoIP protocol. Next, we have developed solutions to mitigate this attack while incurring an acceptable overhead on the one-way path latency. We have presented a brief sketch of the implementation of our proposal on a Phex peer-to-peer client. A detailed experiment evaluation shows that our guards protect caller identity without significantly impacting the quality of voice calls.

## CHAPTER V

### TRUST AND REPUTATION MANAGEMENT

Reputation systems have been popular in estimating the trustworthiness and predicting the future behavior of nodes in a large-scale distributed system where nodes may transact with one another without prior knowledge or experience. One of the fundamental challenges in distributed reputation management is to understand vulnerabilities and develop mechanisms that can minimize the potential damages to a system by malicious nodes. In this chapter, we identify three vulnerabilities that are detrimental to decentralized reputation management and propose TrustGuard – a *safeguard* framework for providing a highly dependable and yet efficient reputation system. First, we provide a dependable trust model and a set of formal methods to handle strategic malicious nodes that continuously change their behavior to gain unfair advantages in the system. Second, a transaction based reputation system must cope with the vulnerability that malicious nodes may misuse the system by flooding feedbacks with fake transactions. Third, but not the least, we identify the importance of filtering out dishonest feedbacks when computing reputation-based trust of a node, including the feedbacks filed by malicious nodes through collusion. Our experiments show that, comparing with existing reputation systems, our framework is highly dependable and effective in countering malicious nodes regarding strategic oscillating behavior, flooding malevolent feedbacks with fake transactions, and dishonest feedbacks.

#### **5.1 Introduction**

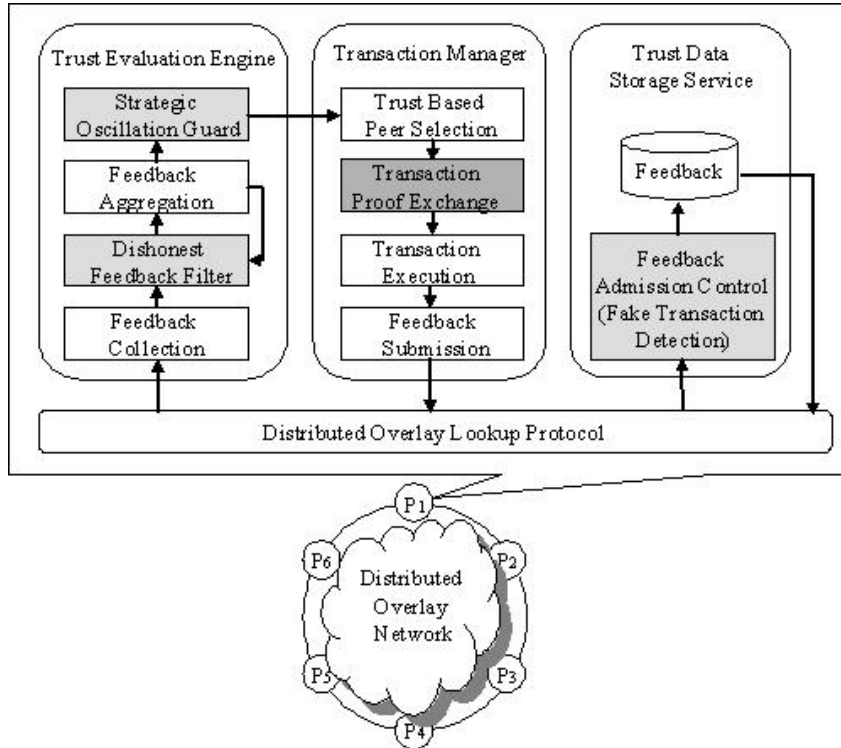
A variety of electronic markets and online communities have reputation system built in, such as eBay, Amazon, Yahoo! Auction, Edeal, Slashdot, Entrepreneur. Recent works [24, 6, 22, 48, 119] suggested reputation based trust systems as an effective way for nodes to identify and avoid malicious nodes in order to minimize the threat and protect the system from possible misuses and abuses by malicious nodes in a decentralized overlay networks.

Such systems typically assign each node a trust value based on the transactions it has performed with others and the feedbacks it has received. For example, XRep [24] provides a protocol complementing current Gnutella protocol by allowing peers to keep track of and share information about the reputation of other peers and resources. EigenTrust [48] presents an algorithm similar to PageRank [68] that computes a trust value by assuming trust is transitive and demonstrated its benefits in addressing fake file downloads in a peer-to-peer file sharing network.

However, few of the reputation management work so far have focused on the vulnerabilities of a reputation system itself. One of the detrimental vulnerabilities is that a malicious node may strategically alter its behavior in a way that benefits itself such as starting to behave maliciously after it attains a high reputation. Another widely recognized vulnerability is the shilling attack [54] where malicious nodes submit dishonest feedback and collude with each other to boost their own ratings or bad-mouth non-malicious nodes. Last, but not the least, malicious nodes can flood numerous fake feedbacks through fake transactions in a transaction-based feedback system.

We believe that a highly dependable reputation system is needed to safeguard the system itself from malicious attacks through strategic oscillation, fake transactions, and dishonest feedbacks. For example, a node's reputation (represented by its trust value) should drop quickly as soon as it misbehaves [123], while it should be hard for any node to boost its reputation within a short period of time. In addition, the trust building techniques based on reputation should be robust and effective in countering attacks through fake transactions and dishonest feedbacks.

With these issues in mind, we present TrustGuard – a highly dependable reputation-based trust building framework. The chapter has a number of unique contributions. First, we introduce a highly dependable trust model to effectively handle strategic oscillations by malicious nodes (Section 5.3). Second, we propose a feedback admission control mechanism to ensure that only transactions with secure proofs can be used to file feedbacks (Section 5.4). Third, we propose feedback credibility based algorithms for effectively filtering out dishonest feedbacks (Section 5.5). We describe performance enhancements by selective



**Figure 86:** TrustGuard Architecture

replication and caching in Section 5.6. We also present a set of simulation based experiments, showing the effectiveness of the TrustGuard approach in guarding against each of the above vulnerabilities with minimal overhead in Section 5.7. Our experiments show that, comparing with existing reputation systems, the TrustGuard framework is highly dependable and effective in countering malicious nodes regarding strategic colluding or oscillating behavior, flooding malevolent feedbacks with fake transactions, and using dishonest feedbacks to manipulate ratings. We conclude the chapter with a brief overview of the related work (Section 5.8), and a conclusion (Section 5.9).

## 5.2 *TrustGuard: An Overview*

### 5.2.1 System Architecture

We first present a high level overview of the TrustGuard framework. Figure 86 shows a sketch of the decentralized architecture of the dependable reputation management system. The callout shows that each node has a transaction manager, a trust evaluation engine and

a feedback data storage service. Whenever a node  $n$  wants to transact with another node  $m$ , it calls the *Trust Evaluation Engine* to perform a trust evaluation of node  $m$ . It collects feedback about node  $m$  from the network through the overlay protocol and aggregates them into a trust value. Such computation is guarded by strategic oscillation guard and dishonest feedback filters. The *Transaction Manager* consists of four components. The trust-based node selection component uses the trust value output from the trust evaluation engine to make trust decisions before calling the transaction execution component. Before performing a transaction, the transaction proof exchange component is responsible for generating and exchanging transaction proofs. Once the transaction is completed, the feedbacks are manually entered by the transacting users. The transacting nodes then route these feedbacks to designated nodes on the overlay network for storage through a decentralized overlay protocol (e.g. DHT based protocol). The designated nodes then invoke their *data storage service* and admit a feedback only if it passes the feedback admission control where fake transactions are detected. The feedback storage service is also responsible for storing reputation and trust data on the overlay network securely, including maintaining replicas for feedbacks and trust values. We build the TrustGuard storage service on top of PeerTrust [119].

Although we implement the TrustGuard framework using a decentralized implementation that distributes the storage and computation of the trust values of the nodes, it is important to note that one could implement TrustGuard using different degrees of centralization. At one extremity, third-party trusted servers could be used for both trust evaluation and feedback storage. One can also utilize the trusted servers to support only selected functionality, for example, the transaction proof exchange (Section 5.4).

Finally, we assume that TrustGuard architecture is built on top of a secure overlay network. Thus, the overlay network should be capable of routing messages despite the presence of some malicious nodes and ensure that all nodes can be identified through some digital certification based mechanism disallow malicious nodes from spoofing fake identities. As shown by Douceur in his Sybil attack chapter [30], bad nodes may potentially amplify their strength by a factor that is proportional to the number of identities they can spoof simultaneously. One practical way to counter the pseudo-spoofing attack is to tie an identity

to a node through digital certification based mechanisms or enforce a secure login procedure for nodes wanting to join the overlay network. Readers may refer to [17, 97, 30] for a detailed discussion on security issues in overlay networks.

### 5.2.2 Problem Statement and Solution Approach

The TrustGuard framework is equipped with several important safeguard components. In the rest of the chapter, we focus on the following three types of vulnerabilities, analyze the potential threats and describe countermeasures against such vulnerabilities using TrustGuard.

**Strategic Oscillation Guard.** Most existing reputation systems such as eBay use a combination of average feedbacks and the number of transactions performed by a node as indicators of its trust value. Our experiments show that using a simple average does not guard the reputation system against oscillating behavior or dishonest feedbacks. For example, a bad node may behave non-maliciously until it attains a good reputation (reflected in its trust value) and then behave maliciously. Or it could oscillate between building and milking reputation. A dependable reputation system should be able to penalize malicious nodes for such dynamic and strategic behavioral changes. In TrustGuard, we promote the incorporation of the reputation history and behavior fluctuations of nodes into the estimation of their trustworthiness. We use adaptive parameters to allow different weighting functions to be applied to current reputation, reputation history, and reputation fluctuations. The current reputation of a node is computed from aggregating feedbacks about this node over the recent period. The reputation history is computed with an incremental aggregation of past reputation values of a node with a bias on recent history. The reputation fluctuation is computed using a derivative of the reputation value. We also develop a fading memories based optimization technique to reduce the cost of maintaining historical information of nodes.

**Fake Transaction Detection.** In a typical transaction-based feedback system, after each transaction, the two participating nodes have an opportunity to submit feedbacks about each other. This brings two vulnerabilities. First, a malicious node may flood numerous

ratings on another node with fake transactions. Second, a malicious node may submit dishonest feedback about a transaction. A dependable trust model should be equipped with mechanisms to handle malicious manipulation of feedbacks to guard the system against such fake transactions, and to differentiate dishonest feedback from honest ones. In TrustGuard approach, we propose to bind a feedback to a transaction through transaction proofs. In other words, a feedback between nodes  $n$  and  $m$  on a given transaction is stored *if and only if*  $n$  and  $m$  indeed transacted with each other. Concretely, before two nodes perform a transaction, they exchange an unforgeable transaction proof. Once the transaction proof is fairly exchanged, both the nodes can submit feedback about the transaction using the proof. The feedback is then routed to appropriate nodes for storage and only those feedbacks with valid transaction proof will be admitted into the feedback database.

**Dishonest Feedback Filter.** While the fake transaction detection guarantees that a feedback is associated with a real transaction, a malicious node may submit dishonest feedbacks in order to boost the ratings of other malicious nodes or bad-mouth non-malicious nodes. The situation is made much worse when a group of malicious nodes make collusive attempts to manipulate the ratings. In this chapter, we build a dishonest feedback filter to differentiate dishonest feedbacks from honest ones. The filter essentially assigns a credibility value to a feedback source and weights a feedback in proportion with its credibility. We study two such credibility measures and their effectiveness in filtering out dishonest feedbacks in both non-collusive and collusive settings.

### 5.3 *Strategic Malicious Nodes*

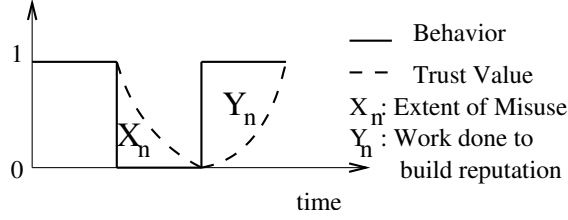
We define a strategic malicious node as a node that adapts its behavioral pattern (with time) so as to maximize its malicious goals. Consider a scenario wherein a bad node does not misbehave until it earns a high trust value. The scenario becomes more complicated when bad nodes decide to alternate between good and bad behavior at regular or arbitrary frequencies. In this chapter, we primarily focus on *strategic oscillations* by malicious nodes and describe concrete and systematic techniques taken by TrustGuard to address both steady and sudden changes in the behavioral pattern of a node without adding heavy

overheads to the system. Other possible behavioral strategies that could be employed by malicious nodes are not considered in this chapter.

A dependable trust model should be capable of handling the following four important issues: (*P1*) sudden fluctuations in node behavior, (*P2*) distinguish an increase and decrease in node behavior, (*P3*) tolerate unintentional errors, and (*P4*) reflect consistent node behavior. We propose a dependable trust model that computes reputation-based trust of a node by taking into consideration: current feedback reports about the node, its historical reputation, and the fluctuations in the node’s current behavior. First, we present an optimization theory based cost metric (Section 5.3.1) to formalize our design goals and then present TrustGuard’s dependable trust model (Section 5.3.2).

### 5.3.1 Cost Model

The primary goal of our safeguard techniques is to maximize the cost that the malicious nodes have to pay in order to gain advantage of the trust system. We first formally define the behavior of a non-malicious and a malicious node in the system using the game theory approach [26]. A non-malicious node is the commitment type and a long-run player who would consistently behave well, because cooperation is the action that maximizes the player’s lifetime payoffs. In contrast a strategic malicious node corresponds to an opportunistic player who cheats whenever it is advantageous for him to do so. Now we formally describe a cost model for building reputation-based trust and use this cost model to illustrate the basic ideas of maximizing the cost (penalty) to be paid by anyone behaving maliciously. Let  $TV_n(t)$  denote the trust value as evaluated by the system for node  $n$  at time  $t$  ( $0 \leq TV_n(t) \leq 1$ ). Let  $BH_n(t)$  denote the actual behavior of node  $n$  at time  $t$  ( $0 \leq BH_n(t) \leq 1$ ), modeled as the fraction of transactions that would be honestly executed by node  $n$  between an infinitesimally small time interval  $t$  and  $t + dt$ . We define the extent to which a malicious node may misuse its reputation (denoted by  $X_n(t)$ ) and the amount of work to be done at time  $t$  by a malicious node in order to increase its reputation-based trust value (denoted by  $Y_n(t)$ ) as  $X_n(t) = \max(TV_n(t) - BH_n(t), 0)$  and  $Y_n(t) = \max(BH_n(t) - TV_n(t), 0)$  respectively. Then, we define the cost function for a node



**Figure 87:** Cost of Building Reputation

$b$  as shown in Equation 16.

$$cost(b) = \lim_{t \rightarrow \infty} \frac{1}{t} * \int_0^t (BH_b(x) - TV_b(x)) dx \quad (16)$$

Let  $G$  be the set of good nodes and  $B$  be the set of bad nodes. The objective is  $\forall g \in G : TV_g(t) \approx 1$  and  $\forall b \in B : cost(b)$  is maximized. Figure 87 provides an intuitive illustration of the above cost function for a strategic malicious node oscillating between acting good and bad. Referring to Figure 87, observe that the problem of maximizing the  $cost$  paid by the malicious nodes can be reduced to maximizing the area under  $Y_n(t) - X_n(t)$ , that is, minimizing the extent of misuse ( $X_n(t) = \max(TV_n(t) - BH_n(t), 0)$ ) and maximizing the cost of building reputation ( $Y_n(t) = \max(BH_n(t) - TV_n(t), 0)$ ).

In addition to maximizing the cost metric, we require TrustGuard to ensure that any node behaving well for an extended period of time attains a good reputation. However, we should ensure that the cost of increasing a node's reputation depends on the extent to which the node misbehaved in the past. For example a node that misbehaved for an extended period of time in the past should find it very hard to build reputation in a short span of time.

### 5.3.2 Dependable Trust Model

Bearing the above analysis in mind, we present TrustGuard's dependable trust model in this section. Let  $R_n(t)$  denote the raw trust value of node  $n$  at time  $t$ . Any of the existing trust evaluation mechanisms such as [119, 48] can be used to calculate  $R_n(t)$ . The simplest form can be an average of the ratings over the recent period of time. Let  $TV_n(t)$  denote the dependable trust value of node  $n$  at time  $t$  and we compute  $TV_n(t)$  using Equation 17.



**Incorporating feedbacks by computing  $R[i]$ .** Let  $R[i]$  denote the raw reputation value of node  $n$  computed as an aggregation of the feedbacks received by node  $n$  in interval  $i$ . Let us for now assume that all the feedbacks in the system are honest and transactions are not faked. In such a scenario,  $R[i]$  can be computed by using a simple average over all the feedback ratings received by node  $n$  in time interval  $i$ . We defer the extension of our safeguard to handle dishonest feedbacks and fake transactions later to sections 5.4 and 5.5 respectively.

**Incorporating History by Computing Integral.** We now compute the integral (history) component of the trust value of node  $n$  at interval  $i$ , denoted as  $H[i]$ . Suppose the system stores the trust value of node  $n$  over the last  $maxH$  (maximum history) intervals,  $H[i]$  could be derived as a weighted sum over the last  $maxH$  reputation values of node  $n$  using Equation 18.

$$H[i] = \sum_{k=1}^{maxH} R[i-k] * \frac{w_k}{\sum_{k=1}^{maxH} w_k} \quad (18)$$

The weights  $w_k$  could be chosen either *optimistically* or *pessimistically*. An example of an optimistic summarization is the exponentially weighted sum, that is,  $w_k = \rho^{k-1}$  (typically,  $\rho < 1$ ). Note that choosing  $\rho = 1$  is equivalent to  $H$  being the average of the past  $maxH$  reputation values of node  $n$ . Also, with  $\rho < 1$ ,  $H$  gives more importance to the more recent reputation values of node  $n$ . We consider these evaluations of  $H$  optimistic since they allow nodes to attain higher trust values rather quickly. On the contrary, a pessimistic estimate of  $H$  could be obtained with  $w_k = \frac{1}{R[i-k]}$ . Such an evaluation assigns more importance to those intervals where the node behaved particularly badly.

**Strengthening the dependability of  $TV[i]$ .** Once we have calculated the feedback-based reputation ( $R[i]$ ) for the node  $n$  in the interval  $i$  and its past reputation history ( $H[i]$ ), we can use Equation 19 to compute the derivative component ( $D[i]$ ). Note that Equation 19 uses  $H[i]$  instead of  $R[i-1]$  for stability reasons.

$$D[i] = R[i] - H[i] \quad (19)$$

Using Equation 19 to approximate the derivative component  $D_n[i]$  has a number of advantages. First, one may use a simple approach to compute the derivative component:

$D_n[i] = R_n[i] - R_n[i - 1]$ . However, we observed that this measure is noisy due to the fact that it is too coarse to use the difference between  $R_n[i]$  and  $R_n[i - 1]$  to approximate the behavioral fluctuations of node  $n$  in the past history. One solution to dampen this noise is to use the standard deviation of  $R_n[i]$  over all intervals in the past history using the formula [44]:  $D_n[i] = \sqrt{\frac{\sum_{i=1}^{maxH} (R_n[i] - \mu)^2}{maxH}}$  where  $\mu = \frac{\sum_{i=1}^{maxH} R_n[i]}{maxH}$ . Note that  $\mu = \frac{\sum_{i=1}^{maxH} R_n[i]}{maxH}$  is the average reputation value over the last  $maxH$  number of intervals. The main concern of computing the standard deviation is the high overhead involved. Therefore, in the first prototype of TrustGuard we choose to use  $D_n[i] = R_n[i] - H_n[i]$  for dampening the noise and reducing the overhead (since history information  $H_n[i]$  is available from the second component in our trust model).

We now compute the dependable trust value  $TV[i]$  for node  $n$  in the interval  $i$  using Equation 20:

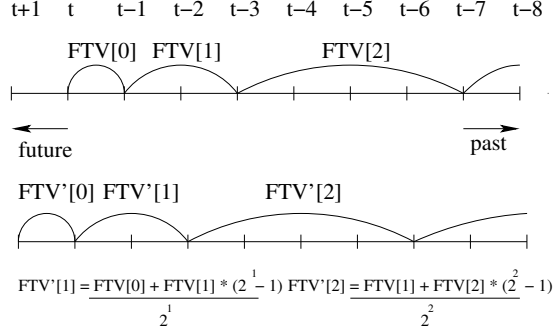
$$TV[i] = \alpha * R[i] + \beta * H[i] + \gamma(D[i]) * D[i]$$

where  $\gamma(x) = \gamma_1$  if  $x \geq 0$  and  $\gamma(x) = \gamma_2$  if  $x < 0$  (20)

In this equation,  $TV[i]$  is derived by associating different weights  $\gamma_1$  and  $\gamma_2$  for a positive gradient and a negative gradient of the trust value respectively, enhancing the dependability of  $TV[i]$  with respect to sudden behavioral changes of node  $n$ . One of the main motivations in doing so is to set  $\gamma_1 < \beta < \gamma_2$ , thereby increasing the strength of the derivative component (with respect to the integral component) when a node shows a fast degradation of its behavior, and lowering the strength of the derivative component when a node is building up its reputation (recall  $P2$  in our design goal). Our experiments (see Section 5.7) show that one can use the rich set of tunable parameters provided by Equation 20 to handle both steady and sudden changes in the behavior of a strategic malicious node.

### 5.3.3 Fading Memories

In TrustGuard, we compute the dependable trust value of a node  $n$  in interval  $i$  based on its current reputation, its reputation history prior to interval  $i$  and its reputation fluctuation. In computing reputation history, we assume that the system stores the reputation-based



**Figure 89:** Updating Fading Memories:  $FTV[i]$  denotes the faded values at time  $t$  and  $FTV'[i]$  denotes the faded values at time  $t + 1$

trust values of node  $n$  for the past  $maxH$  number of intervals. By using a smaller value for  $maxH$ , we potentially let the wrong-doings by a malicious node to be *forgotten* in approximately  $maxH$  time intervals. However, using a very large value for  $maxH$  may not be a feasible solution for at least two reasons: (i) The number of trust values held on behalf of a long standing member of the system could become extremely large. (ii) The computation time for our trust model (Equations 18 and 20) increases with the amount of data to be processed. In the first prototype of TrustGuard, we introduce *fading memories* as a performance optimization technique to reduce the space and time complexity of computing  $TV[i]$  by allowing a trade-off between the history size and the precision of the historical reputation estimate.

One simple technique to trade-off history size with precision would be to aggregate the trust value in each  $k$  consecutive intervals into one value. However, from our experiments we observed that it is vital to keep the trust values in the recent past very precise. Therefore, we propose to aggregate data over intervals of exponentially increasing length in the past  $\{k^0, k^1, \dots, k^{m-1}\}$  into  $m$  values (for some integer  $k > 0$ ). Observe that the aggregates in the recent past are taken over a smaller number of intervals and are hence more precise. This permits the system to maintain more detailed information about the recent trust values of node  $n$  and retain *fading memories* (less detailed) about the older trust values of node  $n$ . Given a fixed value to the system-defined parameter  $m$ , one can trade-off the precision and the history size by adjusting the value of  $k$ .

Now we describe how we implement fading memories in TrustGuard. To simplify the

discussion, let us assume that  $k = 2$ . With fading memory optimization, our goal is to summarize the last  $2^m - 1$  ( $\sum_{i=0}^{m-1} 2^i = 2^m - 1$ ) trust values of a node by maintaining just  $m$  ( $=\log_2(2^m)$ ) values. This can be done in two steps. (i) we need a mechanism to aggregate  $2^m - 1$  trust values into  $m$  values, and (ii) we need a mechanism to update these  $m$  values after each interval.

TrustGuard performs Step 1 as follows. In the interval  $t$ , the system maintains trust values in intervals  $t-1, t-2, \dots, t-2^m$  in the form of  $m$  trust values by summarizing intervals  $t - 2^j, t - 2^j - 1, \dots, t - 2^{j+1} + 1$  for every  $j$  ( $j = 0, 1, \dots, m - 1$ ), instead of maintaining one trust value for each of the  $2^m - 1$  time intervals. Figure 89 provides an example where  $k = 2$  and  $m = 3$ . The upper part of Figure 89 shows that at time  $t$ ,  $FTV[0]$  is a summary over the interval  $\{t\}$ ,  $FTV[1]$  is a summary over the next two intervals  $\{t - 1, t - 2\}$ , and  $FTV[2]$  is a summary over the next four intervals  $\{t - 3, t - 4, t - 5, t - 6\}$ .

Now we discuss how TrustGuard performs Step 2. Let  $FTV^t[j]$  ( $0 \leq j \leq m - 1$ ) denote the faded trust values of node  $n$  at interval  $t$ . Ideally, re-computing  $FTV$  for interval  $t$  requires all of the past  $2^m - 1$  trust values. With fading memories we only store  $m$  summarization values instead of all the  $2^m - 1$  trust values. Thus, at interval  $t$  we approximate the trust value for an interval  $t - i$  ( $1 \leq i \leq 2^m$ ) by  $FTV^t[\lceil \log_2 i \rceil]$ . We use Equation 21 to approximate the updates to the faded trust values for interval  $j$  ( $j = 0, 1, 2, \dots, m - 1$ ) with the base case  $FTV^{t+1}[0] = R[t]$ . Figure 89 gives a graphical illustration of Equation 21 for  $m = 3$ .

$$FTV^{t+1}[j] = \frac{(FTV^t[j] * (2^j - 1) + FTV^t[j - 1])}{2^j} \quad (21)$$

In summary, the fading memories approach closely resembles the way human beings remember their experiences. Our experiments show the vital role of fading memories in optimizing the performance of the trust system while maintaining its dependability at trust model level.

## 5.4 Fake Transactions

We have presented a dependable trust metric, focusing on incorporating reputation history and reputation fluctuation to guard a reputation system from strategic oscillation of malicious nodes. We dedicate this and the next section to vulnerabilities due to fake transactions

and dishonest feedbacks and their TrustGuard countermeasures.

In TrustGuard, we tackle the problem of fake transactions by having a feedback bound to a transaction through a transaction proof such that a feedback can be successfully filed only if the node filing the feedback can show the proof of the transaction. Our transaction proofs satisfy the following properties: (i) Transaction proofs are unforgeable, and are hence generated only if the transacting nodes indeed wished to transact with each other, and (ii) Transaction proofs are always exchanged atomically; that is, a malicious node  $m$  cannot obtain a proof from a non-malicious node  $n$  without sending its own proof to node  $n$ . The atomicity property of the exchange of proofs guarantees fairness; that is, each of the transacting parties would be able to file feedbacks at the end of the transaction. In the absence of exchange atomicity a malicious node  $m$  could obtain a proof from node  $n$  but not provide its proof to the non-malicious node  $n$ ; hence, a non-malicious node  $n$  may *never* be able to file complaints against the malicious node  $m$ . Note that if the entire system is managed by a centralized trusted authority (like eBay) then one can completely eliminate the problem of fake transactions. Our focus is on building a decentralized solution to handle fake transactions.

We first present a technique to generate unforgeable proofs that curb a malicious node from flooding feedbacks on other non-malicious nodes. Then we employ techniques based on electronic fair-exchange protocol to ensure that transaction proofs are exchanged fairly (atomically). It is important to note that the proofs act as signed contracts and are thus exchanged before the actual transaction takes place. If the exchange fails, a good node would not perform the transaction. Nonetheless, if the exchange were unfair, a bad node could file a feedback for a transaction that *never actually* happened.

Note that the fake transaction detection does **not** prevent two collusive malicious nodes from faking a large number of transactions between each other, and further give good ratings with exchanged transaction proofs. This type of collusion will be handled by our next safeguard - dishonest feedback filter. Additionally, one could also deploy a mechanism which associates a non-negligible monetary cost with each transaction so that the attackers may not be able to afford many fake transactions.

### 5.4.1 Unforgeable Transaction Proofs

A simple and secure way to construct proofs of transactions is to use a public key cryptography based scheme. Assume that every node  $n$  has an associated pair of public key and a private key pair, namely,  $\langle PK_n, RK_n \rangle$ . We assume that the public keys are tied to nodes using digital certificates that are notarized by trusted certification authorities. A transaction  $T$  is defined as  $T = \langle Txn Descr \rangle \parallel \langle time stamp \rangle$ , where  $\langle Txn Descr \rangle$  is a description of the transaction and the symbol  $\parallel$  denotes string concatenation. Node  $n$  signs the transaction with its private key to generate a transaction proof  $PT = RK_n(T)$  and send it to node  $m$ . If the proofs are fairly exchanged then node  $n$  would obtain  $RK_m(T)$  as a proof of transaction  $T$  with node  $m$  and vice versa. The key challenge now is how to exchange proofs atomically to guarantee fairness.

### 5.4.2 Fair Exchange of Transaction Proofs

Significant work has been done in the field of fair electronic exchange [78, 62], aiming at guaranteeing *exchange atomicity*. There are several trade-offs involved in employing a fair-exchange protocol in the context of reputation management. In this section we analyze the feasibility of trust value based fair-exchange protocol and optimistic fair-exchange protocol for TrustGuard.

**Trust Value based Fair-Exchange Protocol.** Intuitively, one could achieve fair exchange of proofs between nodes  $n$  and  $m$  ( $TV_n > TV_m$ ) by enforcing that the lower trust value node  $m$  sends its proof first to the higher trust value node  $n$ ; following which the higher trust value node  $n$  would send its proof to the lower trust value node  $m$ . However, this solution is flawed. For example, a malicious node  $m$  with a high trust value may always obtain a proof from non-malicious node  $n$  with a lower trust value, but not deliver its proof to node  $n$ . Hence, a malicious node may pursue its malicious activities *indefinitely* without being *detected* by the trust system.

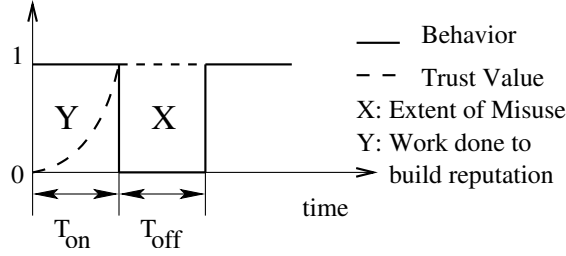
**Optimistic Fair-Exchange Protocol.** In the first prototype of TrustGuard, we adopt an optimistic fair-exchange protocol for exchanging transaction proofs. Optimistic fair-exchange protocols guarantee fair-exchange of two electronic items between two mutually

distrusting parties by utilizing trusted third parties (*ttp*). However, they reduce the involvement of a *ttp* to only those exchanges that result in a *conflict*. Assuming that most of the parties in an open electronic commerce environment are good, the *ttp* is hopefully involved infrequently.

In particular, TrustGuard adopts the optimistic protocol for fair contract signing proposed by Micali [62]. The protocol assumes that the transacting parties  $n$  and  $m$  have already negotiated a would-be contract  $C$ . The nodes  $n$  and  $m$  now need to exchange the signed contracts,  $(RK_n(C)$  and  $RK_m(C))$  fairly. The protocol guarantees that if both the nodes commit to the contract then node  $n$  has a proof that node  $m$  has committed to the contract  $C$  and vice-versa; even if one of the parties does not commit to the contract  $C$  then neither party gets any proof of commitment from the other party. We map this protocol for fairly exchanging transaction proofs by using contract  $C$  as  $C = T = \langle Txn Descr \rangle \parallel \langle time stamp \rangle$ .

Note that minimizing the amount of time the *ttp* needs to be online significantly lowers its susceptibility to attackers. On the other hand, if the *ttp* comes up online very infrequently, the number of outstanding conflicts and the mean time to resolve a conflict would increase significantly. Nonetheless, all conflicts would be eventually resolved.

One of the major advantages of using such an optimistic fair-exchange protocol is that the *ttp* need not be always online. The *ttp* can infrequently come up online and resolve all outstanding conflicts before going offline. A strategic malicious node could exploit the delay in conflict resolution as shown in Figure 90. Let us assume that the *ttp* stays online for a time period  $T_{on}$  and then stays offline for a time period  $T_{off}$ . When the malicious node is building reputation, it behaves honestly and exchanges transaction proofs fairly ( $Y$  in Figure 90). However, after the malicious node has attained high reputation, it unfairly exchanges several proofs with other nodes in the system. By synchronizing the schedule of the *ttp*, the malicious node can ensure that none of the conflicts caused by its malicious behavior is resolved within  $T_{off}$  time units ( $X$  in Figure 90). Hence, despite of the fact that the malicious node behaved badly over a period of  $T_{off}$  time units its reputation does not fall. However, the moment all outstanding conflicts are resolved, the malicious node's



**Figure 90:** Cost of Building Reputation with Delayed Conflict Resolution

reputation falls very steeply. Observe that the cost paid by a malicious node (see Equation 16) is much lower in Figure 90 when compared to Figure 87 (wherein  $T_{off} = 0$ ).

Yet another issue with a large  $T_{off}$  is that the cost of detecting a replay attack on the transaction proofs become expensive. A simple algorithm to detect duplicates would proceed as follows: The trust management system maintains a cache of all proofs submitted on behalf of complaints in the last  $T_{cache}$  time units, where  $T_{cache}$  is an upper bound on the time it takes for a node to file a complaint once it has a proof in hand. When a new proof is submitted on behalf of a complaint, it is accepted only if the time stamp on the proof is no more than  $T_{cache}$  time units behind the current time and if the proof is not found amongst the cache of all proofs submitted in the last  $T_{cache}$  time units. If one were to delay the resolution of conflicts, one would have increase  $T_{cache}$  to  $T_{cache} + T_{off}$ . Hence, the size of the cache maintained by the trust management system would blow up by a factor  $1 + \frac{T_{off}}{T_{cache}}$ .

In conclusion, one needs to choose  $T_{off}$  very carefully so that: (i) the attackers do not have enough time to compromise the trusted third party, and (ii) maximize the cost paid by those malicious nodes that strategically exploit delayed conflict resolution.

## 5.5 Dishonest Feedbacks

In the previous section we have discussed techniques to ensure that both transacting nodes have a fair chance to submit feedbacks. In this section we extend our safeguard model to handle dishonest feedbacks. The goal of guarding from dishonest feedbacks is to develop algorithms that can effectively filter out dishonest feedbacks filed by malicious nodes in the system.

We propose to use a credibility factor as a filter in estimating the reputation-based trust

value of a node in the presence of dishonest feedbacks. Recall that we use  $TV_n$  to denote the dependable trust value of node  $n$  and  $R_n$  to denote the reputation-based trust value of node  $n$  without incorporating past history (Integral component) and fluctuations (Derivative component). The main idea of using a credibility-based feedback filter in computing  $R_n$  is to assign higher weight to the credible feedbacks about node  $n$  and lower weight to the dishonest ones.

Concretely, we first extend the naive average based computation of trust value (Section 5.3.2) into a weighted average. Let  $I(n)$  denote the set of interactions (transactions) performed by node  $n$ . Let  $F_n(u)$  denote the normalized feedback rating (between 0 and 1) that a node  $n$  receives after performing an interaction  $u$  with another node. Let  $CR_n(u)$  denote the feedback credibility of the node  $u.x$  who submitted the feedback about node  $n$  after interaction  $u$ . The reputation-based trust of node  $n$  can be computed as  $R_n = \sum_{u \in I(n)} F_n(u) * CR_n(u)$ . The information about the set of transactions performed ( $I(n)$ ) and the feedbacks received ( $F_n(u)$  for  $u \in I(n)$ ) can be collected automatically [119]. Our goal is to design a credibility filter function that is most effective in ensuring that more credible feedbacks are weighted higher and vice-versa.

A simple and intuitive solution is to measure feedback credibility of a node  $n$  using its trust value  $TV_n$ . We call it the Trust-Value based credibility Measure (TVM for short). Let  $TV_{u.x}$  denote the trust value of node  $u.x$  who had interaction  $u$  with node  $n$ . We can compute the trust value based credibility measure of node  $u.x$  in the interaction  $u$ , denoted by  $CR_n^{TVM}(u)$ , using Equation 22.

$$CR_n^{TVM}(u) = \frac{TV_{u.x}}{\sum_{u \in I(n)} TV_{u.x}} \quad (22)$$

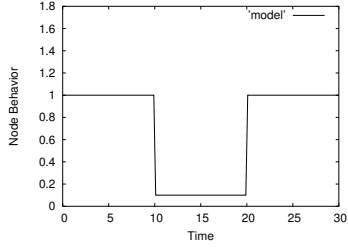
Several existing reputation-based trust systems use TVM or its variant to measure feedback credibility [119, 48]. The TVM solution is based on two fundamental assumptions. First, untrustworthy nodes are more likely to submit false or misleading feedbacks in order to hide their own malicious behavior. Second, trustworthy nodes are more likely to be honest on the feedback they provide. It is widely recognized that the first assumption is generally true but the second assumption may not be true. For example, it is possible that a node may

maintain a good reputation by providing high quality services but send malicious feedbacks to its competitors. This motivates us to design a more effective credibility measure.

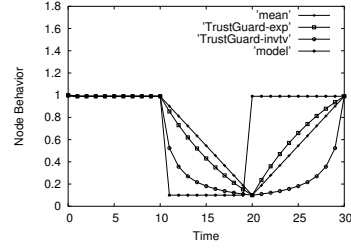
We propose to use a *personalized similarity measure* (PSM for short) to rate the feedback credibility of another node  $x$  through node  $n$ 's personalized experience. Concretely, a node  $n$  will use a personalized similarity between itself and node  $x$  to weigh all the feedbacks filed by node  $x$  on any other node (say  $y$ ) in the system. Let  $IJS(n, x)$  denote the set of common nodes with whom both node  $n$  and  $x$  have interacted, and  $I(n, r)$  denotes the collection of interactions between node  $n$  and node  $r$ . We compute similarity between node  $n$  and  $x$  based on the root mean square of the differences in their feedback over the nodes in  $IJS(n, x)$ . More specifically, given a node  $m$  and an interaction  $u \in I(m)$  performed by node  $m$  with node  $u.x$ , node  $n$  computes a personalized similarity-based credibility factor for  $u$ , denoted as  $CR_n^{PSM}(u)$ , using Equation 23.

$$\begin{aligned}
 CR_n^{PSM}(u) &= \frac{Sim(n, u.x)}{\sum_{u \in I(n)} Sim(n, u.x)} \text{ where} & (23) \\
 Sim(n, x) &= 1 - \sqrt{\frac{\sum_{r \in IJS(n, x)} (A(n, r) - A(x, r))^2}{|IJS(n, x)|}} \\
 A(n, r) &= \frac{\sum_{v \in I(n, r)} F_n(v)}{|I(n, r)|}
 \end{aligned}$$

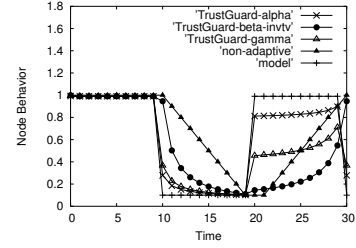
This notion of personalized (local) credibility measure provides a great deal of flexibility and stronger predictive value as the feedback from similar raters are given more weight. It also acts as an effective defense against potential malicious cliques of nodes that only give good ratings within the clique and give bad rating outside the clique. Using personalized credibility to weight the feedbacks will result in a low credibility for dishonest feedbacks by malicious cliques. This is particularly true when measuring the feedback similarity between a node  $m$  in a clique and a node  $n$  outside the clique. Our experiments show that PSM outperforms TVM when the percentage of malicious nodes become large and when the malicious nodes collude with each other.



**Figure 91:** Model I



**Figure 92:** Optimistic versus Pessimistic Summarization



**Figure 93:** Effect of Varying Parameters in the Trust Model

## 5.6 Performance Enhancements

We have so far proposed techniques to guard from strategic node behaviors, dishonest and fake transactions. However for the trust system to be feasible its performance should be satisfactory. In this section, we provide some guidelines to enhance the performance of a distributed trust system assuming that in the fraction of non-malicious nodes *always* tend to dominate the system.

### 5.6.1 Efficiently Storing Feedbacks

Assuming that most nodes are non-malicious and our trust system functions correctly in assigning trust values to nodes, one would expect that a large majority of the transaction in the system turn out successful. When two nodes  $n$  and  $m$  interact, node  $n$  (symmetrically node  $m$ ) uses the following strategy for sending its feedback about the interaction. If the interaction were to turn out successful, node  $n$  would send its feedback directly to node  $m$ . It is in the interest of node  $m$  to retain the feedback and present it to other nodes to prove its goodness. However, if the interaction turns out unsuccessful (with a small probability), node  $n$  sends its feedback to a set of  $R$  nodes that hold *complaints* against node  $m$ . Note that node  $m$  cannot be trusted to store a negative feedback, since node  $m$  may as well delete the feedback (manipulating the feedback is not possible because the feedback is digitally signed). Hence, when a node  $k$  is interested in the trust value of node  $m$ , it can obtain all positive feedbacks about node  $m$  directly from node  $m$  and obtain all negative feedbacks about node  $m$  from the set of  $R$  replica nodes. Suppose node  $m$  were good a large fraction

$f$  of its interactions would receive positive feedbacks and hence reduces the communication cost for the querying node  $k$  by a fraction  $f * (1 - \frac{1}{R})$  and the average storage space required per transaction to  $1 - f + R * f$ .

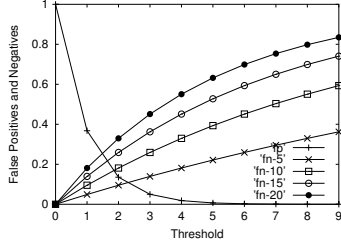
### 5.6.2 Minimizing the Replica Lookup Cost

Any node  $n$  would typically query the trust value of a couple of other nodes before deciding to interaction with one of them. In doing so, node  $n$  would have to locate the  $R$  feedback holders of node  $m$ . Distributed reputation management systems like [119, 48] use a DHT-based structure [106, 77] to locate feedback holders on an overlay network. Typically, the feedback holders for node  $m$  is derived as nodes responsible for  $h(ID(m) || i), 1 \leq i \leq R$ , where  $ID(m)$  is usually  $h(IP(m))$  ( $IP(m)$  is IP-address of node  $m$ )<sup>1</sup>. Hence, node  $n$  would have to perform a lookup for each of these  $R$  identifiers to locate the feedback holders of node  $m$ . This would cost node  $n$  about  $O(\log N) * R$  hops on the overlay network. We can potentially avoid this cost by ensuring that every node  $m$  maintains a cache of its feedback holders. This cache at node  $m$  could be updated by notifications when a feedback holder of node  $m$  enters or leaves the system. This would significantly cut down the *online* communication cost for a node  $n$  wanting to file a negative feedback about node  $m$  since it almost entirely avoids the lookup cost for locating feedback nodes.

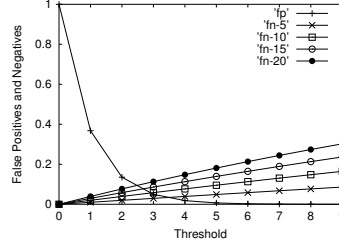
However, in doing so, we potentially run into the danger of node  $m$  providing a list of its own *friends* as its feedback holders. Hence, effectively none of the negative feedbacks filed against node  $m$  would be recorded in the system. We can circumvent this problem by using certain properties of the DHT-system. For example, node  $n$  knows that the  $i^{th}$  feedback holder of node  $m$  is a node that is responsible for the key  $K_i = h(ID(m) || i)$ . Hence, if node  $m$  claims that its  $i^{th}$  feedback holder is some node  $q$ , node  $n$  can verify if the key  $K_i$  is *reasonably close* to  $ID(q) = h(IP(q))$ . Clearly, this scheme has scope for both *false positives* and *false negatives*. A false positive occurs when a node  $n$  mistakenly believes that node  $q$  cannot be the node that is responsible for key  $K_i$ , when node  $q$  is actually responsible for key  $K_i$ . A false negative occurs when a node  $n$  mistakenly believes

---

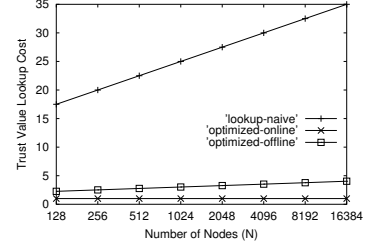
<sup>1</sup> $h$  is some standard hash function like MD5 [80] or SHA1 [32]



**Figure 94:** Probability of False Positives and False Negatives with 100% Collusion



**Figure 95:** Probability of False Positives and False Negatives with 20% Collusion



**Figure 96:** Trust Value Lookup Cost

that a node  $q$  is responsible for key  $K_i$ , when node  $q$  is actually not responsible for key  $K_i$ . One can show that given the distance threshold ( $x_{thr}$ ) and the fraction of malicious nodes known to any malicious node  $m$  ( $p_m$ ) one can show that the false positive ( $FP$ ) and the false negative ( $FN$ ) probabilities are as follows:

$$FP(x_{thr}) = e^{-x_{thr}} \quad (24)$$

$$FN(x_{thr}, p_m) = 1 - e^{-x_{thr} * p_m} \quad (25)$$

For further details refer to the Appendix. Observe that the probability of false negative increases with the threshold value. Figure 94 and 95 shows the probability of false positives and false negatives for different values of threshold ( $x_{thr}$ ) and the fraction of malicious nodes in the system ( $p$ ). Figure 94 shows the results for  $p_m = p$ , that is, node  $m$  is aware of all the bad nodes in the system. Figure 95 shows the results for a more realistic (less pessimistic) scenario wherein we assume that only 20% of the malicious nodes collude ( $p_m = \frac{p}{5}$ ). Observe from Figure 95 that for  $x_{thr} = 2.5$ , the false positive and the false negative probabilities is under 0.1 even for  $p = 20\%$ .

Figure 96 shows the latency in terms of the number of application level hops required to lookup the trust value of one node in the system using the *naive* and the *optimized* techniques. Recall that the naive technique performance a DHT-lookup, one for each feedback holder of the node. In the optimized technique the node whose trust value is being queried itself returns its feedback holders. Hence the *online* cost of locating a nodes feedback holders is just one hop. However, this technique requires that each node maintains a cache of

its feedback holders. We used a simple update based caching mechanism wherein if a node  $m$  becomes a feedback holder for node  $n$ , then it updates node  $n$ 's cache. We assumed that the mean transaction rate ( $\lambda$ ) is about ten times the mean life time ( $\frac{1}{\mu}$ ) of a node [120] (equivalently, an average node performs ten transactions before failure). Say the mean query rate  $\lambda = 1$  per node per second and each node queries the trust value of 10 other nodes before choosing to interact with one of them the naive scheme would cost it about 350 hops/node/second while the optimized scheme costs online 10 hops/node/second and offline 0.4 hops/node/second (Note,  $\mu = \frac{\lambda}{10} = 0.1$ ). In spite of the fact that the optimized technique has scope for false positives and negatives (see Figures 94 and 95) and possibilities of stale/outdated cache entries (less than 10% in our experiments), its performance benefits motivates us to employ it for efficiently looking up trust values.

## 5.7 Evaluation

In this section, we report results from our simulation based experiments to evaluate TrustGuard's approach to build dependable reputation management. We implemented our simulator using a discrete event simulation [36] model. Our system comprises of about  $N = 1024$  nodes; a random  $p\%$  of them is chosen to behave maliciously. In the following portions of this section, we demonstrate the effectiveness of the three guards that we have proposed in this chapter.

### 5.7.1 Guarding from Strategic Node Behaviors

In this section we evaluate the ability of our trust model to handle dynamic node behaviors. We first study the behavior of our guard against strategic oscillations by comparing the optimistic and pessimistic summarization techniques. We demonstrate the significance of various parameters in our dependable trust metrics by varying the weights assigned to reports received in the recent time window ( $\alpha$ ), the history ( $\beta$ ), and the derivative component ( $\gamma$ ). Then, we show the impact of history size ( $maxH$ ) on the effectiveness of our trust model and the advantages of storing past experiences using fading memories.

For all experiments reported in this section, we studied four different models of strategic

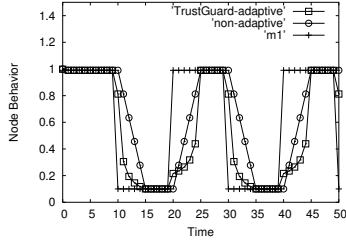
malicious behaviors (refer Section 5.3.1 for the definition of node behavior). In Model I shown in Figure 91, the malicious nodes oscillate from good to bad behavior at intervals of regular time periods. In model II, the malicious nodes oscillate between good and bad behaviors at exponentially distributed intervals. In model III, the malicious nodes choose a random level of goodness and stay that level for an exponentially distributed duration of time. In model IV the malicious node shows a sinusoidal change in its behavior that is the node steadily and continuously changes its behavior unlike models I, II and III which show sudden fluctuations.

#### 5.7.1.1 Comparing Optimistic and Pessimistic Summarizations

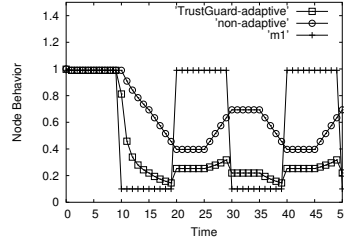
We first compare the two types of weighted summarization techniques discussed in Section 5.3.2. Figure 92 shows the values obtained on summarization given the node behavior model I shown in Figure 91 (using  $\rho = 0.7$ ,  $maxH = 10$  and time period of malicious behavior oscillation = 10). The result shows that *mean value (mean)* and *exponentially weighted sum (exp)* have similar effect and they both are more optimistic than the *inverse trust value weighted sum (invtv)*. Observe that the more pessimistic a summarization is, the harder it is for a node to attain a high trust value in a short span of time and the easier it is to drop its trust value very quickly. Also observe that the exponentially weighted sum in comparison to the mean rises quite steeply making it unsuitable for summarization.

#### 5.7.1.2 Trust Model Parameters

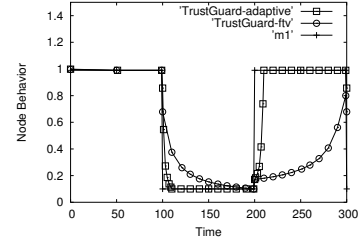
Figure 93 shows the results obtained from our trust model with various parameter settings under the malicious behavior shown in model I (*m1*). *alpha* shows the results obtained when  $\alpha$  is the dominant parameter ( $\alpha \gg \beta, \gamma$ ). With a dominant  $\alpha$  the trust model almost follows the actual behavior of the node since it amounts to disregarding the history or the current fluctuations in the behavior of the node (see Equation 17). *beta-*invtv** shows the results obtained with  $\beta$  as the dominant parameter using inverse trust value weighted sum. With more importance given to the behavior history of a node, the trust value of a node does not change very quickly. Instead it slowly and steadily adapts to its actual behavior. *gamma* shows the results obtained with  $\gamma$  being the dominant factor. With a large  $\gamma$  the



**Figure 97:** Trust Model with a Small History



**Figure 98:** Trust Model with a Large History



**Figure 99:** Trust Model with Fading Memories

trust value responds very swiftly to sudden changes in the behavior of the node. Observe the steep jumps in the trust value that correspond to the time instants when the node changes its behavior. These results match our intuition, namely,  $\alpha$ ,  $\beta$  and  $\gamma$  are indeed the weights attached to the current behavior, historical behavior and the fluctuations in a node's behavior. Finally, *non-adaptive* shows the trust value of a node in the absence of dependable schemes to handle dynamic node behaviors. From Figure 93 it is evident that the cost paid by a malicious node in a non-adaptive model is almost zero, while that in a dependable model is quite significant. A more concrete evaluation that considers the combined effect of various trust model parameters is a part of our ongoing work.

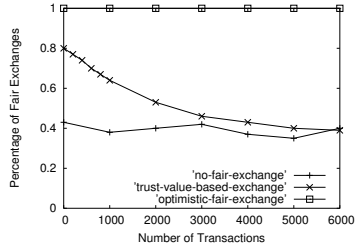
### 5.7.1.3 Varying History Size

In this section we show the effect of history size  $maxH$  on the *cost* (see Equation 16) paid by malicious nodes. Figure 97 shows a scenario wherein the malicious nodes oscillate in their behavior every 10 time units. Note that in this experiment we used  $\alpha = 0.2$ ,  $\beta = 0.8$ ,  $\gamma_1 = 0.05$  and  $\gamma_2 = 0.2$ . Based on our experiences with the dependable trust model one needs to choose  $\alpha$  and  $\beta$  such that  $\frac{\beta}{\alpha}$  is comparable to  $maxH$  (intuitively, this weights the history component in proportion to its size ( $maxH$ )). Note that this experiment uses  $maxH = 5$  which is less than the time period of oscillations by the malicious nodes. From Figure 97 it is clear that the dependable trust models (*TrustGuard-adaptive* in figure) performs better in terms of cost to be paid by the malicious nodes than the non-adaptive trust model (recall the cost model in Section 5.3.1). However, this does not entirely maximize the cost paid by malicious nodes. Figure 98 shows the trust values obtained when  $\alpha = 0.1$ ,  $\beta = 0.9$ ,

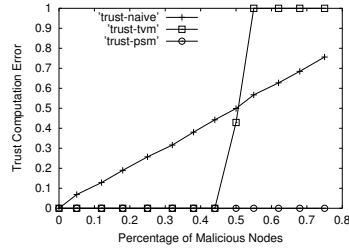
$\gamma_1 = 0.05$ ,  $\gamma_2 = 0.2$  and  $maxH = 15$  (larger than the time period of oscillation by the malicious node). Clearly, having a larger history ensures that one can maximize the cost paid by the malicious nodes. In fact, one observes that the cost paid by malicious nodes for  $maxH$  equal to 5, 10 and 15 are in the ratio of 0.63 : 1 : 3.02 respectively. This observation tells us that if a strategic malicious node knew that  $maxH = 5$ , then it would oscillate at a period equal to 5 time intervals since anyway the system does not remember its past performance beyond 5 time intervals. In short, by knowing the exact value of  $maxH$ , a strategic malicious node would start to oscillate with time period equal to  $maxH$  so as to minimize its cost. It is interesting to note that, when the non-adaptive model is used, the cost paid by malicious nodes is close to zero for all values of time period of behavior oscillation and history size  $maxH$ .

#### 5.7.1.4 Fading Memories

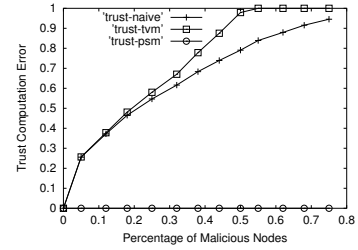
We now evaluate the effectiveness of the fading memories technique in efficiently storing the performance of a node over the last  $2^{maxH}$  intervals using a logarithmically small number of values. Figure 99 shows the effect of using fading memories when a malicious node oscillates with time period equal to 100 time units. It compares a dependable trust model (*TrustGuard-adaptive* in figure) with  $maxH = 10$  and a dependable trust model using fading memories (*TrustGuard-ftv* in figure) based technique with  $m = 8$ . From Figure 99 it is apparent that using a simple adaptive technique with  $maxH = 10$  enables a bad node to recover from its bad behavior that stretched over 100 time units in just 10 additional time units, since the past performance of the node is simply forgotten after 10 time units. As we discussed in Section 5.3, one of the design principles for dependable trust management is to prevent a bad node that has performed poorly over an extended period of time to attain a high trust value quickly. Clearly, the adaptive fading memories based technique can perform really well in this regard, since using just 8 values, it can record the performance of the node over its last 256 ( $2^8$ ) time intervals. It is important to note that the solution based on fading memories has bounded effectiveness in the sense that by setting  $m = 8$ , any node could erase its malicious past over 256 time intervals. However, the key benefit of our fading



**Figure 100:** Fair Exchange of Transaction Proofs: Optimistic Vs Trust-Value Based Exchange Protocol



**Figure 101:** Robustness in Non-Collusive Setting



**Figure 102:** Robustness in Collusive Setting

memories based approach is its ability to increase the cost paid by malicious nodes, with minimal overhead on the system performance.

#### 5.7.1.5 Other Strategic Oscillation Models

We also studied the cost of building reputation under different bad node behavior models discussed in the beginning of Section 5.7.1. From our experiments, we observed that the response of our trust model towards models II, III and IV are functionally identical to that obtained from model I (Figure 91). However, from an adversarial point of view, we observed that these strategies do not aid in minimizing the cost to be paid by malicious nodes to gain a good reputation when compared to model I. In fact, the cost paid by malicious nodes using models I, II, III and IV are in the ratio of 1 : 2.28 : 2.08 : 1.36. In models II and III, the malicious nodes do not pursue their malicious activities the very moment they attain a high reputation. In model IV, the malicious nodes slowly degrade their behavior, which does not given them good benefits (see the extent of misuse  $X_n(t)$  in Figure 87) when compared to a steep/sudden fall. Hence, a strategic malicious node that is aware of  $maxH$  would oscillate with time period  $maxH$  in order to minimize its cost (refer Equation 16). Nonetheless this emphasizes the goodness of our dependable trust model since it is capable of effectively handling even its worst vulnerability (model I with oscillation time period  $maxH$ ).

## 5.7.2 Guarding from Fake Transactions

In this section we study the feasibility of using optimistic fair-exchange protocol for exchanging transaction proofs. First, we demonstrate the superiority of optimistic fair-exchange protocol over the trust value based exchange protocol. Second, we show the effect of varying the server offline duration on the effectiveness of the trust management system.

### 5.7.2.1 Trust Value Based Protocol Vs Optimistic Protocol

Figure 100 shows the percentage of fair exchange of transaction proofs with progress in time for the two exchange protocols discussed in Section 5.4, namely the trust value based fair exchange protocol and the optimistic fair exchange protocol. The experiment measures the percentage of fair transactions when 20% of the nodes are malicious. The trust value based exchange scheme suffers because a strategic malicious node may gain high trust value initially and then fake arbitrarily large number of transactions by unfairly exchanging transaction proofs without being *detected* by the system.

### 5.7.2.2 Trusted Server Offline Duration in Optimistic Protocol

As we have discussed in Section 5.4, it is important to decrease the amount of time the trusted third party server is online so as to make it less susceptible to attackers. However, doing so increases the amount of time it takes to resolve a conflict. We have shown in Section 5.4.2 that a malicious node can exploit the delay in conflict resolution to may ensure that none of its malicious activities be made visible to the trust management system for  $T_{off}$  units of time. In this experiment, we show how the transaction success rate varies with  $T_{off}$ , the time period for which the trusted third party server is offline.

Table 18 shows the normalized cost (see Equation 16) paid by malicious nodes when we introduce a delay in conflict resolution. We assume that all good nodes file complaints as soon as they have the transaction proof that enables them to do so. Recall that a malicious node may ensure that none of its malicious activities be made visible to the trust management system for  $T_{off}$  units of time. We have normalized  $T_{off}$  with the history size ( $maxH$ ) maintained by TrustGuard’s adaptive trust model (see Section 5.3). Figure 18

$T_{off}$	0	0.05	0.1	0.2	0.3
$Cost$	1	0.9	0.85	0.78	0.66

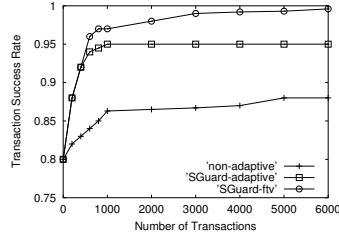
**Table 18:** Relative Cost paid by Malicious Nodes Vs  $T_{off}$  (normalized by  $maxH$ )

shows that in order to keep the cost from dropping more than 10%,  $T_{off}$  should be no more than 5% of  $maxH$ . Note that this is another scenario where fading memories (see Section 5.3) helps the system. Fading memories essentially allow the history size ( $maxH$ ) to be very large and hence the duration of the time for which a trusted third party server is offline could be sufficiently large without significantly decreasing the cost paid by malicious nodes.

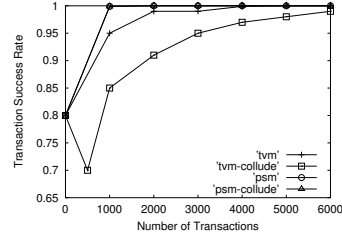
### 5.7.3 Guarding from Dishonest Feedbacks

In this section we present an evaluation of our algorithm to filter dishonest feedbacks (Section 5.5). Recall that the fake transaction guard does not prevent fake transactions between two malicious nodes. So, we simulated two settings, namely, non-collusive and collusive setting. In the collusive setting, a group of collusive malicious nodes may attempt to deterministically boost their ratings by providing highly positive feedbacks on each other through innumerable fake transactions.

Figures 101 and 102 show the error in trust computation as a function of the fraction of malicious nodes in the system in a non-collusive and a collusive setting respectively. Observe that the naive technique (an average of the feedback without credibility factor) for computing trust drops almost linearly with fraction of malicious nodes. Also, the naive technique and the TVM approach are extremely sensitive to collusive attempts even when the number of malicious nodes is very small. On the other hand, the PSM approach remains effective even with both large fraction of malicious nodes and collusion. Recall that PSM computes a *personalized* trust value and hence, the trust value of a node may be different from the perspective of various other nodes in the system. For example, the trust value of a node  $m$  from the perspective of other nodes within its clique may be very high, and yet, the trust value of node  $m$  as seen by other nodes in the system might be very low. Therefore, the similarity metric used by PSM is very effective even in an overwhelming presence of collusive malicious nodes.



**Figure 103:** Transaction Success Rate: Assuming No Fake or Dishonest Feedbacks



**Figure 104:** Transaction Success Rate: Non-Collusive and Collusive Settings

### 5.7.4 Transaction Success Rate

In addition to low level metrics like the cost paid by malicious nodes, we also report a higher level metric, namely, the mean *transaction success rate* that can be achieved using a trust based node selection. We say that a transaction is successful if both the participating nodes cooperate. Given that the trust value of all nodes are made available by our trust system, a node  $n$  can use this value to identify one node from a collection of nodes that is *most qualified* to perform a job. For instance, a node may use a simple threshold based technique wherein, a node  $n$  decides to transact with node  $m$  only if  $TV(m) > TV_{thr}(n)$ , where  $TV_{thr}(n)$  is the trust threshold used by node  $n$ .

Figure 103 shows the transaction success rate that can be achieved using a non-adaptive model (*non-adaptive*), a dependable model (*TrustGuard-adaptive*) and a dependable model using fading memories (*TrustGuard-ftv*). A non-adaptive model pays for its inability to adapt the trust value of a node quickly when the node behavior changes. A dependable model with fading memories has a clear edge over the basic dependable model because it encodes an exponentially larger chunk of a node’s past in a given storage space.

Figure 104 shows the transaction success rate in a system consisting of  $N = 1024$  nodes with 20% nodes being malicious under both collusive and non-collusive settings. Observe from Figure 104 that the transaction success rate is close to 100% for both TVM and PSM in spite of their non-zero trust computation error (see Figures 101 and 102). This is because even if the computed trust values are not 100% accurate, they do differentiate good nodes from bad nodes in most cases by their relative ranking.

## 5.8 Discussion

### 5.8.1 Issues

In this chapter we have described the TrustGuard framework for building secure and distributed reputation management system. The TrustGuard framework includes a trusted third party (TTP) to handle fake transactions. However, the TTP could become a performance bottleneck and a single point of failure. Further, if the TTP were compromised by an adversary then the adversary would be able fake transactions in the system. Our experiments show that the TTP does not become a performance bottleneck with 1024 nodes in the system. However it would be interesting to study the performance, fault-tolerance and scalability of the TTP for larger systems.

The TrustGuard framework assumes that it is built on top of a secure overlay network. Thus, the overlay network should be capable of routing messages despite the presence of some malicious nodes and ensure that all nodes can be identified through some digital certification based mechanism disallow malicious nodes from spoofing fake identities. As shown by Douceur in his Sybil attack chapter [30], bad nodes may potentially amplify their strength by a factor that is proportional to the number of identities they can spoof simultaneously. One practical way to counter the pseudo-spoofing attack is to tie an identity to a node through digital certification based mechanisms or enforce a secure login procedure for nodes wanting to join the overlay network. Readers may refer to [17, 97, 30] for a detailed discussion on security issues in overlay networks.

In the PSM based trust evaluation model, relationship between nodes can be very sparse. Hence, it might get hard to find sufficient number of raters towards a common target node. The Birthday paradox states that given a set of  $N$  elements, two randomly chosen subsets of size  $\sqrt{N}$  have a common element with probability  $\frac{1}{2}$ . Hence, with  $N = 1024$  nodes, two nodes  $n$  and  $m$  that have interacted with 32 random nodes each, have at least one common node with probability  $\frac{1}{2}$ . Hence, even when relationships between nodes are sparse, one might be able to find raters. Further, one could additionally use a combination of TVM and PSM to alleviate the problem of sparse raters.

### 5.8.2 Related Work

Dellarocas [26] provides a working survey for research in game theory and economics on reputation. The game theory based research lays the foundation for online reputation systems research and provides interesting insights into the complex behavioral dynamics. Most of the game theoretic models assume that stage game outcomes are publicly observable. Online feedback mechanisms, in contrast, rely on private (pair-wise) and subjective ratings, thereby raising concerns on the incentive for providing feedbacks or the truthfulness of the feedback.

Related to reputation systems that help establishing trust among entities based on their past behaviors and feedbacks, there is research on propagating trust among entities based on their trust relationship. Yu and Singh [124] propose using historical trust values to update trust values for centralized reputation management systems. Whitby et. al [116] suggest using statistical filtering techniques to update trust values. Yu and Singh [123] also propose a framework based on a gossip protocol. Richardson et al. [79] developed a path-algebra model for trust propagation. Very recently, Guha et al. [41] developed a formal framework for propagating both trust and distrust. The TrustGuard framework is capable of accommodating these algorithms by replacing its dishonest feedback guard.

In the P2P domain reputation management systems like P2Prep [22], Xrep [24] and EigenTrust [48] have emerged. P2Prep provides a protocol on top of Gnutella to estimate trustworthiness of a node. It does not discuss trust metrics in detail and does not have evaluations. XRep extends P2Prep by assigning a reputation value for both peers and resources. EigenTrust assumes that trust is transitive and addresses the weakness of the assumption and the collusion problem by assuming there are pre-trusted nodes in the system. We argue that pre-trusted nodes may not be available in all cases. More importantly, neither of these reputation management systems addresses the temporal dimension of this problem (strategic behavior by malicious nodes) and the problem of fake transactions.

Dellarocas [27] has shown that storing feedback information on the most recent time interval is enough; and that summarizing feedback information for more than one window

of time interval does not improve the reputation system. However, this result subsumes that there are no errors in the feedbacks and that all nodes behave rationally. In the presence of dishonest feedbacks there are bound to be errors in identifying a honest feedback from a dishonest one. Further, our experiments show that the history component helps in stabilizing the system by avoiding transient fluctuations due to transient errors or dishonest feedbacks.

B. Yu and M. P. Singh [123] suggest refining personal opinions differently for cooperation and defection and achieves a certain level of adaptivity. Our dependable trust model is based upon the PID controller popularly used in control theory [67], as against ad hoc techniques suggested in their chapter.

S. K. Lam and J. Riedl [54] experimentally studied several types of shilling attacks on recommender systems. Our experiments show that TrustGuard is resistant to random shilling attacks. As a part of our future work, we hope to model and analyze different types of shilling attacks on reputation systems and enhance our algorithms to further counter them.

Fair exchange protocols [78, 62, 45] have been the prime focus of researchers working in the field of electronic commerce. Ray and Ray [78] provides a survey on fair exchange of digital products between transacting parties. They compare various algorithms including trusted third parties, true & weak fair exchanges, gradual exchanges and optimistic exchanges. In this chapter, we used an optimistic fair-exchange protocol proposed by Micali [62] for fair-contract signing.

## ***5.9 Summary***

We have presented TrustGuard – a framework for building distributed dependable reputation management systems with the countermeasures against three detrimental vulnerabilities, namely, (i) strategic oscillation guard, (ii) fake transaction guard, and (iii) dishonest feedback guard. In TrustGuard we promote a modular design such that one could add more safeguard components, or replace the techniques for one module without having to worry about the rest of the system. The main contribution of this chapter is threefold. First, we

proposed to measure the trustworthiness of peers based on current reputation, reputation history and reputation fluctuation and develop formal techniques to counter strategic oscillation of malicious nodes. Second, we presented electronic fair-exchange protocol based techniques to rule out the possibility of faking transactions in the system. Third, we developed algorithms to filter out dishonest feedbacks in the presence of collusive malicious nodes. We have demonstrated the effectiveness of these techniques through an extensive set of simulation based experiments. We believe that the TrustGuard approach can efficiently and effectively guard a large-scale distributed reputation system, making it more dependable than other existing reputation-based trust systems.

## CHAPTER VI

### CONCLUSIONS AND FUTURE WORK

With the ever increasing rate of digital information available from online services, fostered by the proliferation of ubiquitous computing devices and pervasive networks, security has increasingly become a requirement for system correctness. Over the last decade, overlay network computing model has emerged as a new computing paradigm for large scale parallel and distributed computing. However, due to the large and growing size of overlay networks and their pervasive nature, they have become increasingly more vulnerable to security threats and attacks. Hence, the key challenge here is to retrofit security properties into legacy systems.

In this thesis we have developed system level security algorithms and techniques to support overlay network based applications. In particular, we focused on security issues in a general overlay network and studied two popular applications: publish/subscribe networks and VoIP networks. While it is widely acknowledged that security large scale distributed systems is a complicated problem, we have analyzed and developed guards to protect such systems against a wide range of attacks. In general we have shown that overlay network computing platforms that exhibit highly decentralized and distributed control, require new techniques and solutions for addressing security and scalability problems. However, a common design philosophy employed in this thesis successfully applies to scalably securing overlay network applications. Concretely, we have demonstrated techniques to retrofit security into legacy applications in the form of small customizable plug-ins in way that we minimally modify the legacy system code and yet preserve its performance and scalability.

In particular, this thesis includes the following four major developments targeted towards developing scalable security solutions in overlay networks, publish/subscribe networks and VoIP networks.

**Overlay Network Security:** We have described LocationGuard – a technique for securing

wide area serverless file sharing systems from targeted file attacks. Analogous to traditional cryptographic keys that hide the contents of a file, LocationGuard hides the location of a file on an overlay network. LocationGuard protects a target file from DoS attacks, host compromise attacks, and file location inference attacks by providing a simple and efficient access control mechanism with minimal performance and storage overhead. The unique characteristics of LocationGuard approach is the careful combination of location key, routing guard, and an extensible package of location inference guards, which makes it very hard for an adversary to infer the location of a target file by either actively or passively observing the overlay network. Our experimental results quantify the overhead of employing location guards and demonstrate the effectiveness of the LocationGuard scheme against DoS attacks, host compromise attacks and various location inference attacks.

We have presented TrustGuard – a framework for building distributed dependable reputation management systems with the countermeasures against three detrimental vulnerabilities, namely, (i) strategic oscillation guard, (ii) fake transaction guard, and (iii) dishonest feedback guard. In TrustGuard we promote a modular design such that one could add more safeguard components, or replace the techniques for one module without having to worry about the rest of the system. The main contribution of this chapter is threefold. First, we proposed to measure the trustworthiness of peers based on current reputation, reputation history and reputation fluctuation and develop formal techniques to counter strategic oscillation of malicious nodes. Second, we presented electronic fair-exchange protocol based techniques to rule out the possibility of faking transactions in the system. Third, we developed algorithms to filter out dishonest feedbacks in the presence of collusive malicious nodes. We have demonstrated the effectiveness of these techniques through an extensive set of simulation based experiments.

**Publish/Subscribe Network Security:** We have presented *EventGuard*, a secure system architecture for protecting pub-sub services from various attacks. EventGuard offers security features that are critical to pub-sub overlay services, such as authenticity, confidentiality, integrity, and resilience to flooding based DoS attacks. We have described the two key components of EventGuard: The first component is a suite of security guards that

secure the basic publish and subscribe operations from DoS attacks and unauthorized reads and writes. These guards can be plugged-into a wide-area content-based pub-sub system in a seamless manner. The second component is a resilient pub-sub network design that is capable of providing secure and yet scalable message routing, countering message dropping-based DoS attacks. A unique feature of EventGuard is its unified security framework that meets both security goal for safeguarding the pub-sub overlay services from various vulnerabilities and threats and performance goal for maintaining the simplicity and scalability of the overall system while providing security guarantees. We have reported a series of experimental evaluations, showing that EventGuard can secure a pub-sub overlay service with minimal performance penalty. Our prototype implementation on top of Siena [16] also demonstrates that EventGuard is easily stackable on any content-based pub-sub core.

**VoIP Network Security:** We have addressed the problem of tracing the caller (and the receiver) in encrypted peer-to-peer VoIP networks. We have developed two attacks: one on the SIP protocol layer and the second on the voice session layer. On the SIP protocol layer, we have developed three triangulation based timing attacks that can identify the caller with high probability. We have shown that these attacks can be implemented by passively observing the search traffic on the VoIP network. We have demonstrated the effectiveness of these attacks using the popular Skype VoIP protocol. Next, we have developed random walk based solutions to mitigate this attack while incurring an acceptable overhead on the one-way path latency. On the voice session layer, we have identified flow analysis attacks that can reveal the identity of a receiver to an external observer. We have used network flowing analysis and statistical inference to study the efficacy of such an attack assuming that some of the VoIP network nodes may be compromised. Second, we have developed mixing based techniques to provide a guaranteed level of anonymity for VoIP clients. We have developed an anonymity aware session initiation protocol (AASIP) that allows clients to specify personalized privacy requirements for their voice calls using a quantifiable  $k$ -anonymity metric. We have presented a brief sketch of the implementation of our proposal on a Phex [1] peer-to-peer client. A detailed experiment evaluation shows that our guards protect caller identity without significantly impacting the quality of voice calls.

## 6.1 *Open Issues and Future Research Directions*

While this thesis has presented a set of techniques, algorithms and systems level security architectures for building large scale overlay network applications, it also draws attention to a number of open issues. In this section, we start with discussing open issues in the context of this thesis and then present an outlook of future directions.

### 6.1.1 **Open Issues in the Context of this Thesis**

First, we present open issues in context of overlay networks and the two applications studied in this thesis: publish/subscribe networks and VoIP networks.

**Overlay Networks:** In the context of overlay network security, we have developed algorithms for secure routing and techniques to hide data objects on an overlay network. Our solutions require that the nodes have unique identities. We ensured this by either using a PKI infrastructure or a login-password mechanism through a trust gateway to the overlay network. However, identity management in a completely open overlay network is an open problem. Recent work on Sybil attack [30] illustrates the hardness of identity management in open systems. As a result studying weak identifiers (that limit the number of pseudo-identities) in the context of open overlay networks is an important extension to this research problem.

The second research problem of interest in this context is the problem of building resilient reputation management systems. In this thesis, we have developed guards to protect a reputation management system from three vulnerabilities: oscillating behavior, fake transactions and dishonest feedbacks. However, these attacks do not cover all possible strategies that could be deployed by malicious nodes to subvert the reputation management system. One potential approach to solve this problem would be to model it as a greedy multi-player game and identify optimal attack and defense strategies.

The third research problem of crucial interest in this context is *algorithmic* DoS attacks. In our work we have studied several flooding and spamming based network attacks. All such attacks can be categorized under network level DoS attacks. Algorithmic DoS attacks target several average case assumptions made by the system (such as Web server

cache, DB operations, thread management), while keeping the attack nearly undetectable at the network level. The subtle nature of algorithmic DoS attacks makes it very hard to exhaustively enumerate them, let alone detect and curb them. While an overlay network may use its distributed infrastructure to protect from network level DoS attacks, mitigating algorithmic DoS attacks in overlay network applications is an open problem.

**Publish/Subscribe Networks:** In this thesis we have proposed secure and scalable access control algorithms for publish/subscribe networks. However, our solutions do not apply to all pub-sub matching operators, although it covers most of the popular ones [16]. One solution is to use computation and communication intensive secure multi-party communication protocols. Nonetheless, scalable access control for arbitrary matching operators remains an open problem.

This thesis has also presented secure and scalable content-based routing protocols for publish/subscribe networks. However, our solution does not provide completely confidentiality on an event's routable attributes. We presented techniques to largely increase the entropy (high entropy = high uncertainty = more confidential) using probabilistic multi-path event routing. Guaranteeing absolutely confidentiality on routable event attributes while supporting scalable content-based routing is a challenging problem.

**VoIP Networks:** In this thesis we have pointed out caller identification attacks on VoIP networks and developed random walk based solution mechanisms to alleviate the problem. As the random walk protocol incurs high one-way path latency, we have proposed hybrid algorithms that combine the good features of the broadcast and the random walk based search protocols. However, identifying a distributed search protocol that is Pareto-optimal with respect to both one-way latency and attack resilience is an open problem.

This thesis has also identified and developed flow analysis attacks on VoIP networks. We have proposed a  $k$ -anonymity based solution to protect the identities of the caller and the receiver from an external observer. However, providing complete privacy ( $k = \text{total number of clients in the network}$ ) for the callers and the receivers without using cover traffic remains a challenging problem. Additionally, in the context of VoIP networks, such a privacy solution must also satisfy the one-way latency constraint.

### 6.1.2 An Outlook for Future Research Directions

**Compositionality of Security Plug-ins:** Compositionality has become a corner stone in software design as more and more applications are built simply by composing one or more components. Hence, studying compositionality of security mechanisms is an intrinsic requirement in building usable security architectures for emerging applications. In the context of security plug-ins, one should ensure that when two or more plug-ins are installed into an application, then the plug-ins do not conflict with one another or interact in a way that results in unintended behavior. We believe that for such security plug-ins to be highly usable, it must support policies that allow two or more of these mechanisms to be composed in different ways. These policies will allow an administrator to achieve an appropriate level of security, while suitably trading off the system's performance and scalability metrics. Therefore, developing expressive and yet usable specification languages and formal reasoning techniques for studying policy based compositionality of security mechanisms is a challenging area for future research.

**Ubiquitous Computing Applications:** We believe that the recent emergence of autonomic and pervasive computing has extended the envelope of the Internet to include: mobile and wireless nodes, automatic resource discovery, self-healing and self-optimizing behavior, etc. However, there are several interesting security challenges in this area due to resource constraints (computing, networking and storage) on the weaker wireless/sensor/mobile nodes. Mobility makes this problem more complicated by making the network topology volatile and thus susceptible to a wide range of DoS attacks. Hence, it is of utmost important to study security issues in this domain using several representative applications and security threats including: location privacy for mobile applications, access control for data stream management applications, highly available resource discovery and routing algorithms for continual query applications, and combating voice spam in VoIP networks. A related problem is to study security issues in such ubiquitous applications assuming that the nodes are built upon small trusted computing base. The use of a trusted computing base may significantly alter the threat model, limit the efficacy of several attacks and make

it feasible to develop highly scalable and efficient ubiquitous computing systems.

**Bridging Fault-Tolerance and Security:** With the increasing growth of massively distributed systems, fault-tolerance and security have become an integral part of a system's correctness requirement. While fault-tolerance studies natural faults in a system, security delves into human induced faults that are intentional and goal driven (say, monetary benefits). Several concepts and algorithms proposed in systems security share commonalities with fault-tolerance. For instance, fault-tolerance techniques such as replication and redundancy tend to improve availability: we have used fault-tolerance techniques to build a  $r$ -resilient event delivery network to defend against message dropping based DoS attacks. The key challenge here is to understand the commonality and differences between the notion of faults in fault-tolerance and that of attacks in security. Also, it would be very challenging to study the applicability of fault-tolerant algorithms in building secure and high performance systems.

**Security: Ground Up Architecture Vs Retrofitting:** In this thesis we have advocated a retrofitting model for injecting security features into a legacy system. Our work has focused on large legacy applications (such as publish/subscribe networks, VoIP networks), identifying and analyzing security vulnerabilities, developing guards in the form of customizable plug-ins that can be neatly weaved into the legacy system code. While we have demonstrated this methodology using two popular applications the key question remains: Is it always feasible to retrofit security mechanisms into a legacy system?

## REFERENCES

- [1] “Phex client.” <http://www.phex.org>.
- [2] “Skype - the global internet telephone company.” <http://www.skype.com>.
- [3] “Telegeography research.” <http://www.telegeography.com>.
- [4] “VoIP watch.” <http://voipwatch.com>.
- [5] “Scalable access control in content-based publish-subscribe systems,” Tech. Rep. GIT-CERCS-06-05, Georgia Institute of Technology, 2006.
- [6] ABERER, K. and DESPOTOVIC, Z., “Managing trust in a peer-2-peer information system,” in *Proceedings of the 10th International Conference of Information and Knowledge Management*, 2001.
- [7] ADYA, A., BOLOSKY, W., CASTRO, M., CERMAK, G., CHAIKEN, R., DOUCEUR, J. R., HOWELL, J., LORCH, J. R., THEIMER, M., and WATTENHOFER, R. P., “Farsite: Federated, available and reliable storage for an incompletely trusted environment,” in *Proceedings of the 5th International Symposium on OSDI*, 2002.
- [8] AGUILERA, K. and STROM, R., “Efficient atomic broadcast using deterministic merge,” in *Proceedings of the 19th ACM PODC*, 2000.
- [9] AGUILERA, M., STROM, R., STURMAN, D., ASTLEY, M., and CHANDRA, T., “Matching events in a content-based subscription system,” in *Proceedings of the 18th ACM PODC*, 1999.
- [10] B. ZHAO, J. K. and JOSEPH, A., “Tapestry: An infrastructure for fault-tolerance wide-area location and routing,” Tech. Rep. UCB/CSD-01-1141, University of California, Berkeley, 2001.
- [11] BACK, A., GOLDBERG, I., and SHOSTACK, A., “Freedom 2.1 security issues and analysis.” Zero Knowledge Systems, Inc. white paper, 2001.
- [12] BANAVAR, G., CHANDRA, T., MUKHERJEE, B., and NAGARAJARAO, J., “An efficient multicast protocol for content-based publish subscribe systems,” in *Proceedings of the 19th ICDCS*, 1999.
- [13] BERNSTEIN, D. J., “SYN cookies.” <http://cr.yp.to/syncookies.html>.
- [14] BLUM, A., SONG, D., and s. VENKATARAMAN, “Detection of interactive stepping stones: Algorithms and confidence bounds,” in *7th RAID*, 2004.
- [15] CARZANIGA, A., “Siena - software.” <http://serl.cs.colorado.edu/~carzanig/siena/software/index.html>
- [16] CARZANIGA, A., ROSENBLUM, D. S., and WOLF, A. L., “Design and evaluation of a wide-area event notification service,” in *ACM Transactions on Computer System*, 19(3):332-383, 2001.

- [17] CASTRO, M., DRUSCHEL, P., GANESH, A., ROWSTRON, A., and WALLACH, D. S., "Secure routing for structured peer-to-peer overlay networks," in *OSDI*, 2002.
- [18] CERT, "Incident note IN-2004-01 W32/Novarg.A virus," 2004.
- [19] CHAUM, D., "Untraceable electronic mail, return addresses, and digital pseudonyms," in *Communications of ACM*, 24(2): 84-88, 1981.
- [20] CNN, "Gates: Buy stamps to send email." <http://www.cnn.com/2004/TECH/internet/03/05/spam.charge.ap/>.
- [21] COHEN, E. and JEFFERSON, D., "Protection in the hydra operating system," in *Proceeding of the ACM Symposium on Operating Systems Principles*, 1975.
- [22] CORNELLI, F., DAMIANI, E., DI VIMERCATI, S. D. C., PARABOSCHI, S., and SAMARATI, P., "Choosing reputable servents in a p2p network," in *Proceedings of the 11th World Wide Web Conference*, 2002.
- [23] DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., and STOICA, I., "Wide-area cooperative storage with cfs," in *Proceedings of the 18th ACM SOSP*, October 2001.
- [24] DAMIANI, E., VIMERCATI, S., PARABOSCHI, S., SAMARATI, P., and VIOLANTE, F., "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *CCS*, 2002.
- [25] DATTA, A. K., GRADINARIU, M., RAYNAL, M., and SIMON, G., "Anonymous publish/subscribe in P2P networks," in *Proceedings of IPDPS*, 2003.
- [26] DELLAROCAS, C., "The digitization of word-of-mouth: Promises and challenges of online reputation mechanism," in *Management Science*, 2003.
- [27] DELLAROCAS, C., "Sanctioning reputation mechanisms in online trading environments with moral hazard," in *MIT Sloan Working Paper No. 4297-03*, 2004.
- [28] DINGLEDINE, R., MATHEWSON, N., and SYVERSON, P., "Tor: The second generation onion router," in *13th USENIX Security Symposium*, 2000.
- [29] DONOHO, D. L., FLESIA, A. G., SHANKAR, U., PAXSON, V., COIT, J., and STANFORD, S., "Multiscale stepping stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay," in *5th RAID*, 2002.
- [30] DOUCEUR, J., "The sybil attack," in *2nd Annual IPTPS Workshop*, 2002.
- [31] DROMS, R., "RFC 2131: Dynamic host configuration protocol." <http://www.faqs.org/rfcs/rfc2131.html>.
- [32] EASTLAKE, D. and JONES, P., "US secure hash algorithm I." <http://www.ietf.org/rfc/rfc3174.txt>, 2001.
- [33] ECLIPSE, "Aspectj compiler." <http://eclipse.org/aspectj>.
- [34] ELGAMAL, T., "A public key cryptosystem and a signature scheme based on discrete logarithm," in *IEEE transactions on information theory*, 31(4): 469-472, 1985.

- [35] FERGUSON, R. and SENIE, D., “RFC 2267: Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing.” <http://www.faqs.org/rfcs/rfc2267.html>, 1998.
- [36] FIPS, “Data encryption standard (DES).” <http://www.itl.nist.gov/fipspubs/fip46-2.htm>.
- [37] FREEDMAN, M. J. and MORRIS, R., “Tarzan: A peer-to-peer anonymizing network layer,” in *9th ACM CCS*, 2002.
- [38] GNUTELLA, “The gnutella home page.” <http://gnutella.wego.com/>.
- [39] GOH, E. J., SHACHAM, H., MODADUGU, N., and BONEH, D., “Sirius: Securing remote untrusted storage,” in *Proceedings of NDSS*, 2003.
- [40] GOLDSCHLAG, D., REED, M., and SYVERSON, P., “Onion routing for anonymous and private internet connections,” in *Communications of ACM, Vol 42(2)*, 1999.
- [41] GUHA, R., KUMAR, R., RAGHAVAN, P., and TOMKINS, A., “Propagation of trust and distrust,” in *Proceedings of the 13th World Wide Web Conference*, 2004.
- [42] GYONGYI, Z., GARCIA-MOLINA, H., and PEDERSON, J., “Combating web spam with trustrank,” in *Proceedings of 30th VLDB Conference*, 2004.
- [43] HELLWEG, E., “When bot nets attack.” MIT Technology Review, September 2004.
- [44] HIGGINS, J. and KELLER-McNULTY, S., “Concepts in probability and stochastic modeling.” Duxbury Press ISBN: 0534231365, 1995.
- [45] HOLGER VOGT, HENNING PAGNIA, F. C. G., “Modular fair exchange protocols for electronic commerce,” in *Annual Computer Security Applications Conference*, 1999.
- [46] HUI, J. Y., “Switching and traffic theory for integrated broadband networks.” Academic Press ISBN: 079239061X, 1990.
- [47] JAEGER, T. and RUBIN, A. D., “Preserving integrity in remote file location and retrieval,” in *Proceedings of NDSS*, 1996.
- [48] KAMVAR, S., SCHLOSSER, M., and GARCIA-MOLINA, H., “Eigenrep: Reputation management in p2p networks.”
- [49] KANDULA, S., KATABI, D., JACOB, M., and BERGER, A., “Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds,” in *Proceedings of 2nd USENIX NSDI*, 2005.
- [50] KAZAA, “Kazaa home page.” <http://www.kazaa.com/>, 2003.
- [51] KEROMYTIS, A., MISRA, V., and RUBENSTEIN, D., “SOS: Secure overlay services,” in *Proceedings of the ACM SIGCOMM*, 2002.
- [52] KRAWCZYK, H., BELLARE, M., and CANETTI, R., “HMAC: Keyed-hashing for message authentication.” <http://www.faqs.org/rfcs/rfc2104.html>.

- [53] KUBIATOWICS, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., and ZHAO, B., "Oceanstore: An architecture for global-scale persistent storage," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.
- [54] LAM, S. K. and RIEDL, J., "Shilling recommender systems for fun and profit," in *Proceedings of the 13th World Wide Web Conference*, 2004.
- [55] LEYDEN, J., "East european gangs in online protection racket." [www.theregister.co.uk/2003/11/12/east-european-gangs-in-online/](http://www.theregister.co.uk/2003/11/12/east-european-gangs-in-online/).
- [56] LIMEWIRE, "Improving gnutella protocol: Protocol analysis and research proposals." [http://www9.limewire.com/download/ivkovic\\_paper.pdf](http://www9.limewire.com/download/ivkovic_paper.pdf), 2002.
- [57] MALKHI, D., RODEH, O., and REITER, M., "Efficient update diffusion in byzantine environments," in *Proceedings of 20th IEEE SRDS*, 2001.
- [58] MATHWORLD, "The caesar cipher." <http://www.mathworld.com>.
- [59] MATHWORLD, "Shannon entropy." <http://mathworld.wolfram.com/Entropy.html>.
- [60] MATHWORLD, "Menger's theorem." <http://mathworld.wolfram.com/MengersTheorem.html>, 2002.
- [61] MATHWORLD, "Network flow." <http://mathworld.wolfram.com/NetworkFlow.html>, 2002.
- [62] MICALI, S., "Simple and fast optimistic protocols for fair electronic exchange," in *The Proceedings of ACM PODC*, 2003.
- [63] NIST, "AES: Advanced encryption standard." <http://csrc.nist.gov/CryptoToolkit/aes/>.
- [64] OPENSLL, "OpenSSL." <http://www.openssl.org/>.
- [65] OPENSLL, "Timing-based attacks on RSA keys." [http://www.openssl.org/news/secadv\\_20030317.txt](http://www.openssl.org/news/secadv_20030317.txt).
- [66] OPYRCHAL, L. and PRAKASH, A., "Secure distribution of events in content-based publish subscribe system," in *Proceedings of the 10th USENIX Security Symposium*, 2001.
- [67] OZBAY, H., "Introduction to feedback control theory." CRC Press Inc.
- [68] PAGE, L., BRIN, S., MOTWANI, R., and WINOGRAD, T., "The pagerank citation ranking: Bringing order to the web," tech. rep., 1998.
- [69] PARK, S., LIU, L., PU, C., SRIVATSA, M., and ZHANG, J., "Resilient trust management for web service integration," in *Proceedings of 3rd Intl Conference on Web Services (ICWS)*, 2005.
- [70] PERNG, G., REITER, M. K., and WANG, C., "M2: Multicasting mixes for efficient and anonymous communication," in *IEEE ICDCS*, 2006.

- [71] PFITZMANN, A., PFITZMANN, B., and WAIDNER, M., "ISDN-MIXes: Untraceable communication with small bandwidth overhead," in *GI/ITG Conference on Communication in Distributed Systems*, 1991.
- [72] PFITZMANN, A. and WAIDNER, M., "Networks without user observability," in *Computers and Security*, 2(6): 158-166.
- [73] POULSEN, K., "FBI busts alleged DDoS mafia." [www.securityfocus.com/news/9411](http://www.securityfocus.com/news/9411), 2004.
- [74] QIN LV, S. R. and SHENKER, S., "Can heterogeneity make gnutella scalable?," in *Proceedings of the first International Workshop on Peer-to-Peer Systems*, 2002.
- [75] RAFAELI, S. and HUTCHISON, D., "A survey of key management for secure group communication," in *Journal of the ACM Computing Surveys*, Vol 35, Issue 3, 2003.
- [76] RAICIU, C. and ROSENBLUM, D. S., "A secure protocol for content-based publish/subscribe systems." [http://www.cs.ucl.ac.uk/staff/C.Raicu/files/secure\\_pubsub.pdf](http://www.cs.ucl.ac.uk/staff/C.Raicu/files/secure_pubsub.pdf).
- [77] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., and SHENKER, S., "A scalable content-addressable network," in *Proceedings of SIGCOMM Annual Conference on Data Communication*, Aug 2001.
- [78] RAY, I. and RAY, I., "Fair exchange in e-commerce," in *ACM SIGEcomm Exchange*, 2001.
- [79] RICHARDSON, M., AGARWAL, R., and DOMINGOS, P., "Trust management for the semantic web," in *Proceedings of International Semantic Web Conference*, 2003.
- [80] RIVEST, R., "The MD5 message-digest algorithm." <http://www.ietf.org/rfc/rfc1321.txt>, 1992.
- [81] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., and SCHOOLER, E., "RFC 3261: SIP: Session initiation protocol," 2002.
- [82] ROSS, S. M., "Stochastic processes." Wiley ISBN: 0471120626, 1995.
- [83] ROSS, S. M., "Introduction to probability models." Academic Press ISBN: 0125980558, 2002.
- [84] ROWSTRON, A. and DRUSCHEL, P., "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Nov 2001.
- [85] RSA-LABORATORIES, "RSA cryptography standard." <http://www.rsasecurity.com/rsalabs/pkcs/>.
- [86] SAROIU, S., GUMMADI, P. K., and GRIBBLE, S. D., "A measurement study of peer-to-peer file sharing systems," Tech. Rep. UW-CSE-01-06-02, University of Washington, 2001.

- [87] SAVAGE, S., WETHERALL, D., KARLIN, A., and ANDERSON, T., "Practical network support for IP traceback," in *Proceedings of ACM SIGCOMM*, 2000.
- [88] SCHULZRINNE, H., CASNER, S., FREDERICK, R., and JACOBSON, V., "RFC 1889: RTP: A transport protocol for real-time applications," 1996.
- [89] SENGAR, H., WIJESSEKERA, D., WANG, H., and S. JAJODIA, T. .
- [90] SHIELDS, C. and LEVINE, B. N., "A protocol for anonymous communication over the internet," in *ACM CCS*, 2000.
- [91] SHMATIKOV, V. and WANG, M. H., "Timing analysis in low latency mix networks: Attacks and defenses," in *11th ESORICS*, 2006.
- [92] SINGH, A. and SRIVATSA, M., "Apoidea: Decentralized p2p web crawling." <http://disl.cc.gatech.edu/Apoidea/>.
- [93] SIT, E. and MORRIS, R., "Security considerations for peer-to-peer distributed hash tables," in *Proceedings of IPTPS*, 2002.
- [94] SONG, D., WAGNER, D., and PERRIG, A., "Practical techniques for searches over encrypted data," in *IEEE S & P Symposium*, 2000.
- [95] SOUND, G. I., "VoIP: Better than PSTN?." <http://www.globalipsound.com/demo/tutorial.php>.
- [96] SRIVATSA, M., GEDIK, B., and LIU, L., "Scaling unstructured peer-to-peer networks with multi-tier capability aware topologies," in *Proceedings of International Conference on Parallel and Distributed Systems ICPADS*, 2004.
- [97] SRIVATSA, M. and LIU, L., "Vulnerabilities and security issues in structured overlay networks: A quantitative analysis," in *To Appear in the Proceedings of Springer Intl Journal on Information Security. An extended abstract of this paper appeared in Proceedings of 20th IEEE Annual Computer Security Applications Conference (ACSAC)*, 2004.
- [98] SRIVATSA, M. and LIU, L., "Eventguard: Secure event notification architecture for publish-subscribe networks," in *Proceedings of 12th ACM Conference on Computer and Communication Security (CCS)*, 2005.
- [99] SRIVATSA, M. and LIU, L., "Locationguard: Countering targeted file attacks using location keys," in *Proceedings of 14th USENIX Security Symposium*, 2005.
- [100] SRIVATSA, M. and LIU, L., "Securing decentralized reputation management using trustguard," in *To Appear in the Proceedings of Journal on Parallel and Distributed Computing (JPDC), special issue on Security in Grid and Distributed Systems. An extended abstract of this paper appeared in the Proceedings of 14th World Wide Web (WWW 2005) Conference*, 2006.
- [101] SRIVATSA, M. and LIU, L., "Privacy in VoIP networks: A k-anonymity approach," tech. rep., Georgia Institute of Technology, 2007.

- [102] SRIVATSA, M. and LIU, L., “Secure event routing in content-based publish-subscribe networks,” in *Proceedings of 27th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2007.
- [103] SRIVATSA, M., LIU, L., and IYENGAR, A., “Caller identification attacks in VoIP networks,” tech. rep., Georgia Institute of Technology, 2007.
- [104] SRIVATSA, M., XIONG, L., and LIU, L., “Xchange: A distributed protocol for electronic fair exchange,” in *Proceedings of 19th IEEE Intl Parallel and Distributed Processing Symposium (IPDPS)*, 2005.
- [105] STANIFORD, S., PAXSON, V., and WEAVER, N., “How to own the internet in your spare time,” in *Proceedings of USENIX Security Symposium*, 2002.
- [106] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M., and BALAKRISHNAN, H., “Chord: A scalable peer-to-peer lookup service for internet applications,” in *Proceedings of SIGCOMM Annual Conference on Data Communication*, August 2001.
- [107] SYVERSON, P., TSUDIK, G., REED, M., and LANDWEHR, C., “Towards an analysis of onion routing security,” 2000.
- [108] TAYLOR, L., “Botnets and botherds.” <http://sfbay-infragard.org>.
- [109] WANG, C., CARZANIGA, A., EVANS, D., and WOLF, A. L., “Security issues and requirements for internet-scale publish-subscribe systems,” in *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002.
- [110] WANG, X., CHEN, S., and JAJODIA, S., “Tracking anonymous peer-to-peer VoIP calls on the internet,” in *12th ACM CCS*, 2005.
- [111] WANG, X. and REEVES, D., “Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays,” in *10th ACM CCS*, 2003.
- [112] WANG, X., REEVES, D., and WU, S., “Inter-packet delay based correlation for tracing encrypted connections through stepping stones,” in *7th ESORICS*, 2002.
- [113] WANG, X. and REITER, M. K., “Defending against denial-of-service attacks with puzzle auctions,” in *Proceedings of IEEE Symposium on Security and Privacy*, 2003.
- [114] WANG, X. Y. and REEVES, D. S., “Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays,” in *ACM CCS*, 2003.
- [115] WATERS, B., JUELS, A., HALDERMAN, A., and FELTEN, E. W., “New client puzzle outsourcing techniques for dos resistance,” in *Proceedings of 11th ACM CCS*, 2004.
- [116] WHITBY, A., JOSANG, A., and INDULSKA, J., “Filtering out unfair ratings in bayesian reputation systems,” in *Proceedings of the Workshop on Trust in Agent Societies, at the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS2004)*, July 2004.
- [117] WIKIPEDIA, “Birthday paradox.” [http://www.wikipedia.org/wiki/Birthday\\_paradox](http://www.wikipedia.org/wiki/Birthday_paradox).
- [118] WONG, C. K., GOUDA, M. G., and LAM, S. S., “Secure group communications using key graphs,” in *IEEE/ACM Transactions on Networking: 8, 1(Feb), 16-30*, 2000.

- [119] XIONG, L. and LIU, L., "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," in *Proceedings of IEEE TKDE, Vol. 16, No. 7*, 2004.
- [120] YANG, B. and GARCIA-MOLINA, H., "Improving search in peer-to-peer networks," in *22nd Conference ICDCS'03*, July 2002.
- [121] YANG, Y. R., LI, X. S., ZHANG, X. B., and LAM, S. S., "Reliable group rekeying: A performance analysis," in *Proceedings of ACM SIGCOMM*, 2001.
- [122] YODA, K. and ETOH, H., "Finding a connection chain for tracing intruders," in *6th ESORICS*, 2000.
- [123] YU, B. and SINGH, M. P., "A social mechanism of reputation management in electronic communities," in *Proceedings of the 4th International Workshop on Cooperative Information Agents*, 2000.
- [124] YU, B., SINGH, M. P., and SYCARA, K., "Developing trust in large-scale peer-to-peer systems," in *Proceedings of the First IEEE Symposium on Multi-Agent Security and Survivability*, 2004.
- [125] ZEGURA, E. W., CALVERT, K., and BHATTACHARJEE, S., "How to model an inter-network," in *Proceedings of IEEE Infocom*, 1996.
- [126] ZHANG, Y. and PAXON, V., "Detecting stepping stones," in *9th USENIX Security Symposium*, 2000.

## VITA



Mudhakar Srivatsa was born in the Bangalore and brought up in Madras, India. He obtained a bachelors degree in Computer Science and Engineering with a minor in Operations Research, from the Indian Institute of Technology (IIT), Madras, India. Subsequently, he joined the Computer Science Ph.D. program at the College of Computing of Georgia Institute of Technology (Atlanta, GA, USA). As a member of the center for experimental research in computer systems (CERCS), he conducted research on various aspects of secure and high performance distributed systems, including VoIP networks, Publish/Subscribe networks, and overlay networks. His research in these projects has resulted in the development of several useful security tools in the form of composable and customizable plug-ins for large scale distributed systems. His work has also been acknowledged in the form of numerous publications that have appeared in various international conferences and journals on security and distributed systems. He has also been a collaborator with the IBM T.J. Watson Research Center. He holds or applied for a number of patents on his work at IBM, dealing with denial of service (DoS) attacks on Web servers and access control on Web service compositions.