

UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Detecting Malicious Routers**

A dissertation submitted in partial satisfaction of the  
requirements for the degree Doctor of Philosophy  
in  
Computer Science

by

Alper Tugay Mizrak

Committee in charge:

Professor Keith Marzullo, Co-Chair  
Professor Stefan Savage, Co-Chair  
Professor Rene L. Cruz  
Professor Ramesh R. Rao  
Professor Geoffrey M. Voelker

2007

©

Alper Tugay Mizrak, 2007

All rights reserved.

The dissertation of Alper Tugay Mızrak is approved, and it is acceptable in quality and form for publication on microfilm:

---

---

---

---

Co-Chair

---

Co-Chair

University of California, San Diego

2007

## EPIGRAPH

The people think of wealth and power as the greatest fate,  
But in this world a spell of health is the best state.

*Sultan Süleyman the Magnificent*

## TABLE OF CONTENTS

	Signature Page . . . . .	iii
	Epigraph . . . . .	iv
	Table of Contents . . . . .	v
	List of Figures . . . . .	viii
	Acknowledgements . . . . .	x
	Vita . . . . .	xii
	Abstract . . . . .	xiv
Chapter 1	Introduction . . . . .	1
	1.1. Background . . . . .	4
	1.1.1. Secure Protocols on the Control Plane . . . . .	5
	1.1.2. Secure Protocols on the Data Plane . . . . .	6
Chapter 2	Problem Space . . . . .	9
	2.1. General System Model and Assumptions . . . . .	9
	2.1.1. Network Model . . . . .	9
	2.1.2. Synchronous Model . . . . .	10
	2.1.3. Good Path Between Correct Routers . . . . .	10
	2.1.4. Good Terminal Routers . . . . .	10
	2.1.5. Cryptographic Tools and Key Distribution . . . . .	11
	2.1.6. Routing . . . . .	12
	2.2. Threat Model . . . . .	14
	2.2.1. Adversarial Capability . . . . .	14
	2.2.2. Number of Faulty Routers . . . . .	16
	2.3. Centralized Failure Detector via Active Replication . . . . .	17
	2.4. Distributed Failure Detector via Traffic Validation . . . . .	18
	2.4.1. Traffic Validation . . . . .	19
	2.4.2. Distributed Detection . . . . .	27
	2.4.3. Response . . . . .	30
Chapter 3	Literature Review . . . . .	32
	3.1. WATCHERS: A distributed network monitoring approach . . . . .	32
	3.2. HSER: Highly secure and efficient routing . . . . .	37
	3.3. HERZBERG: Early detection of message forwarding faults . . . . .	39
	3.4. PacketObituaries: Packet obituaries . . . . .	40

	3.5. AWERBUCH: An on-demand secure routing protocol resilient to Byzantine failures . . . . .	42
	3.6. SecTrace: Secure Traceroute . . . . .	43
	3.7. PERLMAN: Network layer protocols with Byzantine robustness . . . . .	46
	3.8. StealthProbing: Stealth probing . . . . .	48
	3.9. SATS: Secure split assignment trajectory sampling . . . . .	49
	3.10.ACL: Availability centric routing . . . . .	49
	3.11.GOLDBERG: The role of cryptography in network accountability . . . . .	50
	3.12.ZHANG: Secure routing in ad-hoc networks . . . . .	51
Chapter 4	System Model and Specification . . . . .	52
	4.1. System Model . . . . .	52
	4.2. Specification . . . . .	55
	4.2.1. Traffic Validation . . . . .	55
	4.2.2. Distributed Detection . . . . .	57
Chapter 5	Detecting Malicious Routers . . . . .	61
	5.1. Protocol $\Pi_2$ : A Complete, Accurate Protocol with Precision 2 . . . . .	61
	5.1.1. Overhead . . . . .	64
	5.2. Protocol $\Pi_{k+2}$ : A Complete, Accurate Protocol with Precision $k + 2$ . . . . .	66
	5.2.1. Overhead . . . . .	68
	5.3. Fatih: Prototype System . . . . .	70
	5.3.1. System Architecture . . . . .	70
	5.3.2. Experiences . . . . .	74
Chapter 6	Detecting Congestive Losses . . . . .	78
	6.1. Inferring Congestive Loss . . . . .	79
	6.1.1. Static Threshold . . . . .	80
	6.1.2. Traffic Modeling . . . . .	81
	6.1.3. Traffic Measurement . . . . .	83
	6.2. Protocol $\chi$ . . . . .	84
	6.2.1. Traffic Validation . . . . .	84
	6.2.2. Distributed Detection . . . . .	88
	6.2.3. Response . . . . .	90
	6.3. Analysis of Protocol $\chi$ . . . . .	91
	6.3.1. Accuracy and Completeness . . . . .	91
	6.3.2. Traffic Validation Correctness . . . . .	91
	6.3.3. Overhead . . . . .	92
	6.4. Experiences . . . . .	94
	6.4.1. Simulation . . . . .	94
	6.4.2. Network Emulation . . . . .	96
	6.4.3. Protocol $\chi$ vs. Static threshold . . . . .	102
	6.5. Non-deterministic Queuing . . . . .	102

6.5.1. Random Early Detection . . . . .	103
6.5.2. Traffic Validation for RED . . . . .	104
6.5.3. Experiences . . . . .	107
Chapter 7 Analysis of the Protocols . . . . .	112
7.1. Computing Fingerprints . . . . .	112
7.2. State Size . . . . .	113
7.3. Synchronization . . . . .	114
7.4. Issues . . . . .	115
7.4.1. Multipaths . . . . .	115
7.4.2. TTL . . . . .	115
7.4.3. Multicast . . . . .	116
7.4.4. Fragmentation . . . . .	116
Chapter 8 Conclusion . . . . .	118
Appendix A Set Reconciliation Algorithm . . . . .	120
Appendix B Properties of Protocol $\Pi_2$ and Protocol $\Pi_{k+2}$ . . . . .	123
B.1. Basic Theorems . . . . .	123
B.2. Properties of Protocol $\Pi_2$ . . . . .	124
B.3. Properties of Protocol $\Pi_{k+2}$ . . . . .	125
Appendix C Properties of Protocol $\chi$ . . . . .	127
Bibliography . . . . .	130

## LIST OF FIGURES

Figure 2.1	Failure detector via active replica / state machine approach. . . .	17
Figure 2.2	Per router conservation of traffic. . . . .	25
Figure 2.3	Per interface conservation of traffic. . . . .	26
Figure 2.4	Conservation of traffic of a path-segment, end-to-end. . . . .	26
Figure 2.5	Conservation of traffic of a path-segment, along the path-segment.	27
Figure 3.1	WATCHERS: Transit packet byte counters . . . . .	33
Figure 3.2	Failure detector via a traffic validator per router. . . . .	35
Figure 3.3	WATCHERS: Consorting routers . . . . .	36
Figure 3.4	Failure detector via a traffic validator per path-segment nodes. . .	37
Figure 3.5	PacketObituaries in wide area interdomain network. . . . .	41
Figure 3.6	Failure detector via a traffic validator per path-segment ends. . .	43
Figure 3.7	SecTrace: Byzantine faulty router. . . . .	44
Figure 3.8	PERLMAN: Colluding routers. . . . .	48
Figure 5.1	Protocol $\Pi_2$ : A Complete, accurate protocol with precision 2.	63
Figure 5.2	Based on <i>AdjacentFault</i> ( $k$ ); maximum, average and median size of $P_r$ , i.e. the number of path-segments monitored by an individual router in Protocol $\Pi_2$ . . . . .	65
Figure 5.3	Protocol $\Pi_{k+2}$ : A Complete, accurate protocol with precision $k + 2$ . . . . .	66
Figure 5.4	Based on <i>AdjacentFault</i> ( $k$ ); maximum, average and median size of $P_r$ , i.e. the number of path-segments monitored by an individual router in Protocol $\Pi_{k+2}$ . . . . .	69
Figure 5.5	Fatih System Architecture . . . . .	71
Figure 5.6	Abilene network topology. . . . .	74
Figure 5.7	Fatih in progress. . . . .	75
Figure 6.1	Validating the queue of an output interface. . . . .	84
Figure 6.2	Confidence value for single packet loss test. . . . .	86
Figure 6.3	NS simulation Protocol $\chi$ . . . . .	95
Figure 6.4	Simple topology. . . . .	97
Figure 6.5	No attack. . . . .	98
Figure 6.6	Attack 1: <i>Drop 20% of the selected flows</i> . . . . .	100
Figure 6.7	Attack 2: <i>Drop the selected flows when the queue is 90% full</i> . . .	101
Figure 6.8	Attack 3: <i>Drop the selected flows when the queue is 95% full</i> . . .	101
Figure 6.9	Attack 4: <i>Target a host trying to open a connection by dropping SYN packets</i> . . . . .	101
Figure 6.10	A set $n$ packets. Each packet $fp_i$ is associated with a drop probability $p_i$ and the outcome is either transmitted(TX) or dropped(DR) based on the random number generated during the last packet drop. . . . .	105

Figure 6.11	Without attack. . . . .	106
Figure 6.12	Attack 1: <i>Drop the selected flows when the average queue size is above 45,000 bytes.</i> . . . . .	107
Figure 6.13	Attack 2: <i>Drop the selected flows when the average queue size is above 54,000 bytes.</i> . . . . .	108
Figure 6.14	Attack 3: <i>Drop 10% of the selected flows when the average queue size is above 45,000 bytes.</i> . . . . .	109
Figure 6.15	Attack 4: <i>Drop 5% of the selected flows when the average queue size is above 45,000 bytes.</i> . . . . .	110
Figure 6.16	Attack 5: <i>Target a host trying to open a connection by dropping SYN packets.</i> . . . . .	111

## ACKNOWLEDGEMENTS

I would like to thank my parents, Mediha and Ahmet Mızrak, and my brother, Koray Mızrak. They always encouraged and supported me throughout my life. I would also like to thank all my friends and colleagues. I am very grateful to have known each of them.

I am very much indebted to my advisors and dissertation co-chairs: Prof. Stefan Savage and Prof. Keith Marzullo. They have advised me over the course of my Ph.D. study and always demonstrated ingenuity, patience and kindness. I would like to thank Prof. Geoffrey M. Voelker for kindly agreeing to serve on my doctoral committee and for his feedback. I would like to thank other members of my committee, Prof. Rene L. Cruz and Prof. Ramesh R. Rao, for sacrificing their time and agreeing to be in my committee.

The text of this dissertation, in full or in part, is a reprint of the following materials with the full permission of all co-authors of the papers:

- Alper Tugay Mızrak, Keith Marzullo and Stefan Savage, “Detecting Compromised Routers via Packet Forwarding Behavior,” *UCSD Technical Report*, CS2007-0899, June 2007.
- Alper Tugay Mızrak, Keith Marzullo and Stefan Savage, “Detecting Malicious Packet Losses,” *UCSD Technical Report*, CS2007-0889, April 2007.
- Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage, “Detecting and Isolating Malicious Routers,” *IEEE Transactions on Dependable and Secure Computing*, July-September 2006 (Vol. 3, No. 3) pp. 230-244.

The dissertation author was the primary investigator and author of these publications. The co-authors, Prof. Keith Marzullo and Prof. Stefan Savage, directed and supervised the research that forms the basis for this dissertation. The co-author, Yu-Chung Cheng, contributed with the experiment setup.

Parts of Chapter 1 are reprints of the materials as they appear in the *IEEE Transactions on Dependable and Secure Computing*, 2006, by Alper Tugay Mızrak,

Yu-Chung Cheng, Keith Marzullo and Stefan Savage; and UCSD Technical Report, CS2007-0889, 2007, by Alper Tugay Mızrak, Keith Marzullo and Stefan Savage.

Parts of Chapter 2 are reprints of the material as it appears in UCSD Technical Report, CS2007-0899, 2007, by Alper Tugay Mızrak, Keith Marzullo and Stefan Savage.

Parts of Chapter 4, Chapter 5 and Chapter 7 are reprints of the material as it appears in the IEEE Transactions on Dependable and Secure Computing, 2006, by Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage.

Parts of Chapter 6 are reprint of the material as it appears in UCSD Technical Report, CS2007-0889, 2007, by Alper Tugay Mızrak, Keith Marzullo and Stefan Savage.

Appendix B is a reprint of the material as it appears in the IEEE Transactions on Dependable and Secure Computing, 2006, by Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage.

Appendix C is a reprint of the material as it appears in UCSD Technical Report, CS2007-0889, 2007, by Alper Tugay Mızrak, Keith Marzullo and Stefan Savage.

## VITA

2007	Doctor of Philosophy in Computer Science University of California, San Diego
2002	Master of Science in Computer Science University of California, San Diego
2000-2007	Graduate Student, Research Assistant Department of Computer Science and Engineering University of California, San Diego
2000	Bachelors of Science in Computer Engineering Bilkent University, Ankara, Turkey

## PUBLICATIONS

Alper Tugay Mızrak, Keith Marzullo and Stefan Savage, “Detecting Compromised Routers via Packet Forwarding Behavior,” *UCSD Technical Report*, CS2007-0899, June 2007.

Alper Tugay Mızrak, Keith Marzullo and Stefan Savage, “Detecting Malicious Packet Losses,” *UCSD Technical Report*, CS2007-0889, April 2007.

Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage, “Detecting and Isolating Malicious Routers,” *IEEE Transactions on Dependable and Secure Computing*, July-September 2006 (Vol. 3, No. 3) pp. 230-244.

Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage, “Fatih: Detecting and Isolating Malicious Routers,” *The International Conference on Dependable Systems and Networks (DSN 2005)*, Yokohama, Japan, June 2005.

Alper Tugay Mızrak, Keith Marzullo and Stefan Savage, “Brief Announcement: Detecting Malicious Routers,” *Symposium on Principles of Distributed Computing (PODC 2004)*, St. John’s, Newfoundland, Canada, July 2004.

Alper Tugay Mızrak, Keith Marzullo and Stefan Savage, “Fault-Tolerant Forwarding in the Face of Malicious Routers,” *Workshop on the Future Directions in Distributed Computing (FuDiCo 2004)*, Bertinoro, Italy, June 2004.

Alper Tugay Mızrak, Keith Marzullo and Stefan Savage, “Detecting Malicious Routers,” *UCSD Technical Report*, CS2004-0789, May 2004.

Alper Tugay Mızrak, Yu-Chung Cheng, Vineet Kumar and Stefan Savage, “Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup,” *The IEEE Workshop on Internet Applications (WIAPP 2003)*, San Jose, CA, June 2003.

Alper Tugay Mizrak, “Discovering Paths Traversed by Visitors in Web Server Access Logs,” *The International NAISO Congress Information Science Innovations (ISI 2001)*, Dubai, U.A.E., March 2001.

## FIELDS OF STUDY

Computer Networks, Fault-Tolerant Networks, Distributed Systems.

ABSTRACT OF THE DISSERTATION

**Detecting Malicious Routers**

by

Alper Tugay Mizrak

Doctor of Philosophy in Computer Science

University of California, San Diego, 2007

Professor Keith Marzullo, Co-Chair

Professor Stefan Savage, Co-Chair

The Internet is not a safe place. Unsecured hosts can expect to be compromised within minutes of connecting to the Internet and even well-protected hosts may be crippled with denial-of-service attacks. However, while such threats to host systems are widely understood, it is less well appreciated that the network infrastructure itself is subject to constant attack as well. Indeed, through combinations of social engineering and exploitation of weak passwords, attackers have seized control over of thousands of Internet routers. Once a router has been compromised in such a fashion, an attacker may interpose on the traffic stream and manipulate it maliciously to attack others – selectively dropping, modifying, or re-routing packets.

First, we specify this problem of detecting routers with incorrect packet forwarding behavior and we explore the design space of protocols that implement such a detector. We further present two concrete protocols that differ in accuracy, completeness, and overhead – one of which is likely inexpensive enough for practical implementation at scale. We present a prototype system that implements this approach on a PC router and describe our experiences with it. We believe our work is an important step in being able to tolerate attacks on key network infrastructure components.

Unfortunately, it is quite challenging to attribute a missing packet to a malicious action because normal network congestion can produce the same effect. Modern

networks routinely drop packets when the load temporarily exceeds a router's buffering capacity. Previous detection protocols have tried to address this problem using a user-defined threshold. Recently, we have designed, developed and implemented a new compromised router detection protocol that dynamically infers, based on measured traffic rates and buffer sizes, the number of congestive packet losses that will occur. Once the ambiguity from congestion is removed, subsequent packet losses can be attributed to malicious actions. We have tested this protocol in Emulab and have studied its effectiveness in differentiating attacks from legitimate network behavior. We believe this protocol is the first to automatically predict congestion in a systematic manner and is necessary for making any such network fault detection practical.

# Chapter 1

## Introduction

The Internet is not a safe place. Unsecured hosts can expect to be compromised within minutes of connecting to the Internet and even well-protected hosts may be crippled with denial-of-service attacks. However, while such threats to host systems are widely understood, it is less well appreciated that the network infrastructure itself is subject to constant attack as well.

Such attacks are not mere theoretical curiosities, but are actively employed in practice. Attackers have repeatedly demonstrated their ability to compromise routers, through combinations of social engineering and exploitation of weak passwords or latent software vulnerabilities [4, 51, 70]. One network operator recently documented over 5000 compromised routers as well as an underground market for trading access to them [126]. Once a router is compromised, an attacker need not modify the router's code base to exploit its capabilities. Current standard command-line interfaces from vendors such as Cisco and Juniper are sufficiently powerful to drop and delay packets, send copies of packets to a third party, or "divert" packets through a third party and back. In fact, several widely published documents provide a standard cookbook for transparently "tunneling" packets from a compromised router through an arbitrary third-party host and back again – effectively amplifying the attacker's abilities, including arbitrary packet sniffing, injection or modification [37, 123]. Such attacks can be extremely difficult to detect manually, and it can be even harder to isolate which particular router

or group of routers has been compromised. Even more troubling is Mike Lynn's controversial presentation at the 2005 Black Hat Briefings, which demonstrated how Cisco routers can be compromised via simple software vulnerabilities. Once a router has been compromised in such a fashion, an attacker may interpose on the traffic stream and manipulate it maliciously to attack others – selectively dropping, modifying, or re-routing packets.

One approach to this problem is to detect the act of router intrusion as it happens. For example, traditional host-based intrusion detection techniques, such as system call anomaly analysis [67, 58, 36], could be applied to the router environment as well. However, this approach also has the same limitations as in the host environment – once a router is compromised, the detection software is no longer dependable. Indeed, it has become increasingly common for malware to disable anti-virus and IDS products and we see no reason to believe the router environment will be different. Thus, while such attack detection mechanisms are undoubtedly an important defensive element, in this dissertation we start from the assumption that routers may be successfully compromised.

This dissertation addresses part of a simple, yet increasingly important network security problem: how to detect the existence of compromised routers in a network and then remove them from the routing fabric. The root of this problem arises from the key role that routers play in modern packet-switched data networks. To a first approximation, networks can be modeled as a series of point-to-point links connecting pairs of routers to form a directed graph. Since few endpoints are directly connected, data must be forwarded – hop-by-hop – from router to router, towards its ultimate destination. Therefore, if a router is compromised, it stands to reason that an attacker may drop, delay, reorder, corrupt, modify or divert *any* of the packets passing through this router. Such a capability can then be used to deny service to legitimate hosts, to implement ongoing network surveillance, or to provide an efficient man-in-the-middle functionality for attacking end systems.

Given this threat model, the problem of detecting a compromised router falls

to its neighbors: A compromised router can potentially be identified by correct routers when it deviates from exhibiting expected behavior. This overall approach can be broken into three distinct subproblems:

1. Traffic validation. Traffic information is the basis of detecting anomalous behavior: given traffic entering a part of the network, and an expected behavior for the routers in the network (i.e. a known routing configuration), anomalous behavior is detected when the monitored traffic leaving one part of the network differs significantly from what is expected. However, implementing such validation practically can be quite tricky and requires tradeoffs between the overhead of monitoring, communication, and accuracy.
2. Distributed detection. It is impossible for a single router to determine that its neighbor's behavior is anomalous. Thus, detection requires synchronizing a collection of traffic information and distributing the results so that anomalous behavior can be detected by *sets* of correct routers.
3. Response. Once a router, or set of routers, is thought to be faulty, the forwarding tables of correct routers must be changed to avoid using those compromised nodes. In addition, over longer time scales, an appropriate alert must be raised so human forensic experts can respond appropriately.

This dissertation addresses each of these problems in turn. For different threats, we describe a range of appropriate and efficient traffic validation functions. Next we examine how these functions can be used to build an anomalous behavior detector for compromised routers. Finally, we show how this detector can be integrated into link-state routing to automatically isolate network paths demonstrating anomalous behavior.

Several researchers have developed distributed protocols to detect such traffic manipulations, typically by validating that traffic transmitted by one router is received un-modified by another [21, 87]. However, all of these schemes – including our own [87] – struggle in interpreting the *absence* of traffic. While a packet that has been modified in transit represents clear evidence of tampering, a missing packet is inherently ambigu-

ous: it may have been explicitly blocked by a compromised router or it may have been dropped benignly due to network congestion. In fact, modern routers routinely drop packets due to bursts in traffic that exceed a router's buffering capacity, and the widely-used Transmission Control Protocol (TCP) is designed to *cause* such losses as part of its normal congestion control behavior. Thus, existing traffic validation systems must inevitably produce false positives for benign events and/or produce false negatives by failing to report real malicious packet dropping.

We design, develop, and implement a new compromised router detection protocol that dynamically infers the precise number of congestive packet losses that will occur. Once the congestion ambiguity is removed, subsequent packet losses can be safely attributed to malicious actions. We evaluate this protocol in a small experimental network in the Emulab testbed and demonstrate that it is capable of accurately resolving extremely small and fine-grained attacks. We believe this protocol is the first to automatically predict congestion in a systematic manner and is necessary for making any such network fault detection practical.

## 1.1 Background

There are inherently two sets of threats posed by an adversary: (1) An attacker may subvert the *network control plane* and attack by means of the routing protocol. For example, by issuing false routing advertisements, a compromised router may manipulate how other routers view the network topology, and thereby disrupt service globally. (2) An attacker may subvert the *network data plane* and attack by means of the packet forwarding process. By causing the router to violate the forwarding decisions that it should make based on its routing tables, a compromised router may disrupt communication in the network. Once a router has been compromised, an attacker may interpose on the traffic stream and manipulate it maliciously to attack others – selectively dropping, modifying, or re-routing packets.

Next, we present a very brief introduction to the secure protocols on the con-

control plane addressing the first set of threats before we focus solely on the network data plane attacks.

### 1.1.1 Secure Protocols on the Control Plane

The first threat has received, by far, the lion's share of the attention in the research community, perhaps due its potential for catastrophic effects. By issuing false routing advertisements, a compromised router may manipulate other routers' views of the network topology, and thereby disrupt service globally. For example, if a router claims that it is directly connected to all possible destinations, it may become a "black hole" for most traffic in the network. While this problem is by no means solved in practice, there has been significant progress towards this end in the research community, beginning with the seminal work of Perlman. In her PhD thesis [104], Perlman described robust flooding algorithms for delivering the key state across any connected network, and described a means for explicitly signing route advertisements. There have subsequently been a variety of efforts to impart similar guarantees to existing routing protocols with varying levels of cost and protection. Generally, these techniques break down into two categories: approaches based on ensuring the authenticity of route updates and those based on detecting inconsistency between route updates.

Barbir *et al.* [13] present a model of generic threats to routing protocols. They characterize threat sources into two groups:

- Outsiders are adversaries participating illegitimately in the routing protocol. These attackers may attempt to access subverted links or network equipment so that they can observe control messages and/or announce invalid control messages.
- Insiders are authorized components of the network participating legitimately in the routing protocol. However, they misbehave, e.g., a compromised router behaves arbitrarily or a router is misconfigured.

Barbir *et al.* categorize routing threat actions as deliberate exposure, sniffing, traffic analysis, spoofing, falsification(insertion, deletion, substitution, replaying, inter-

ference, overload), and they categorize threat consequences into four types: disclosure, deception, disruption, and usurpation. Finally, they discuss the attack damages, such as network congestion, blackholes, looping, partitions, churn, instability, overcontrol, clogging, starvation, eavesdropping, and so on.

An illustrative, but not comprehensive, list of the secure protocols on the control plane includes:

- Link state routing
  - Wired networks: FIRE [101], JiNao [135, 108, 60, 25], others [92, 26, 45, 46].
  - Wireless networks: SLSP [100].
- Distance vector routing
  - Wired networks: RIP-TP [102], S-RIP [129], others [118, 52].
  - Wireless networks: ARAN [113], SAODV [139], SEAD [53], RIDAN [121].
  - Sensor networks: [85].
- Interdomain routing: S-BGP [65, 66], soBGP [134], Whisper [122], SPV [54], others [117, 143, 41, 55, 142].
- General security issues in routing: [140, 91, 42].

Some recent developments in secure routing protocols can be found in [99, 137, 48, 10].

### 1.1.2 Secure Protocols on the Data Plane

By contrast, the threat posed by subverting the forwarding process has received comparatively little attention until very recent years. This is surprising since, in many ways, this kind of attack presents a wider set of opportunities to the attacker – not only denial-of-service, but also packet sniffing, modification and insertion – and is both

trivial to implement (a few lines typed into a command shell) and difficult to detect. The rest of this dissertation focuses entirely on the problem of malicious forwarding.

We study the following protocols in detail in Chapter 3 Literature Review:

- Section 3.1 WATCHERS: A distributed network monitoring approach
- Section 3.2 HSER: Highly secure and efficient routing
- Section 3.3 HERZBERG: Early detection of message forwarding faults
- Section 3.4 PacketObituaries: Packet obituaries
- Section 3.5 AWERBUCH: An on-demand secure routing protocol resilient to Byzantine
- Section 3.6 SecTrace: Secure Traceroute
- Section 3.7 PERLMAN: Network layer protocols with Byzantine robustness failures
- Section 3.8 StealthProbing: Stealth probing
- Section 3.9 SATS: Secure split assignment trajectory sampling
- Section 3.10 ACL: Availability centric routing
- Section 3.11 GOLDBERG: The role of cryptography in network accountability
- Section 3.12 ZHANG: Secure routing in ad-hoc networks

In Chapter 5, we present our Protocol  $\Pi_2$  and Protocol  $\Pi_{k+2}$  detecting malicious routers. In Chapter 6, we design and develop Protocol  $\chi$  that dynamically infers the number of congestive packet losses.

- Chapter 5 Protocol  $\Pi_{k+2}$  and Protocol  $\Pi_2$ : Detecting and isolating malicious routers
- Chapter 6 Protocol  $\chi$ : Detecting malicious packet losses

## **Acknowledgement**

Parts of Chapter 1 are reprints of the materials as they appear in the IEEE Transactions on Dependable and Secure Computing, 2006, by Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage; and UCSD Technical Report, CS2007-0889, 2007, by Alper Tugay Mızrak, Keith Marzullo and Stefan Savage.

## Chapter 2

# Problem Space

This chapter studies the problem space in detail and presents a general system model on which all of the existing failure detection protocols rely, and then explore the design space, including the design decisions made in these protocols. As such, this chapter provides a background to study those existing failure detection protocols in Chapter 3.

### 2.1 General System Model and Assumptions

It turns out that all of these protocols entail similar requirements, such as a synchronous network model, good terminal routers, the good path property, cryptographic tools and key distribution. In this section we present these requirements and a general system model for all protocols, presented in Chapter 3, that have addressed the problem of detecting compromised nodes attacking the *data plane*.

#### 2.1.1 Network Model

Some of the protocols are designed for hard-wired networks, while the others are designed for wireless networks. Within a network, it is presumed that packets are forwarded in a hop-by-hop fashion from source to destination – each node following the directions of a local forwarding table, which is computed by routing protocols, as

described in Section 2.1.6.

It is assumed that each node has sufficient data processing capability to generate traffic summaries describing the network traffic it is forwarding, and sufficient computational capability to exchange and reconcile summaries with its neighbors. We discuss the issue of overhead further in Chapter 7.

### **2.1.2 Synchronous Model**

Every protocol assumes a synchronous network model of coarsely synchronized clocks and/or bounded message delays. This assumption is required by the protocols in order to decide whether a packet has been delivered within the expected time interval which is determined via timeout mechanism.

### **2.1.3 Good Path Between Correct Routers**

It is assumed that between any two uncompromised routers, there is sufficient path diversity such that the malicious routers do not partition the network. In some sense, this assumption is pedantic since it is impossible to guarantee any network communication across such a partition. Another way to view this constraint is that path diversity between two points in the network is a necessary, but insufficient, condition for tolerating compromised routers. These protocols all propose a mechanism that offers a sufficiency condition in the presence of the necessary diversity condition.

Recently, Teixeira et al. [124] empirically measured path diversity in ISP networks and found that multiple paths between pairs of routers were common. Similarly, many enterprise networks are designed with such diversity, in order to mask the impact of link failures. Consequently, we believe that this assumption is reasonable in practice.

### **2.1.4 Good Terminal Routers**

It should be noted, however, that this diversity usually does not extend to individual hosts on local-area networks; single workstations rarely have multiple paths to

their network infrastructure. In these situations, for fate-sharing reasons, there is little that can be done. If host's access router is compromised, then the host is isolated and there is no routing remedy even if an anomaly is detected; the fate of individual hosts and their access routers are directly intertwined. Moreover, from the standpoint of the network, such traffic *originates* from a compromised router, and therefore cannot demonstrate anomalous forwarding behavior.<sup>1</sup>

To summarize, these protocols are designed to detect anomalies between pairs of *correct* nodes and thus for simplicity it is assumed that a terminal router is not faulty with respect to traffic originating from or being consumed by that router. This assumption is well justified due to the fate-sharing argument and it is accepted by all of the detection protocols.

This assumption is necessary, in order to protect against faulty terminal routers that drop packets they receive from an end-host or packets they should deliver to an end-host. However, it also excludes denial-of-service(DoS) attacks wherein a faulty router introduces bogus traffic claiming that the traffic originates from a legitimate end-host. Of course, standard rate-limit schemes can be applied against these kind of DoS attacks, yet, none of these protocols explicitly address this problem.

### 2.1.5 Cryptographic Tools and Key Distribution

A negative result is presented in [39] proving that any *Byzantine fault detection protocol* requires a key infrastructure, cryptographic operation, and dedicated storage at every node. All of the protocols that consider a compromised node with the ability to alter packets require cryptographic functions [40, 62]. They are primarily for authenticity and integrity<sup>2</sup>:

- Digital signatures, *eg.* DSA [93].
- Message authentication code (MAC), *eg.* HMAC [17], UMAC [19]

---

<sup>1</sup>This issue can be partially mitigated by extending our protocol to include hosts as well as routers, but this simply pushes the problem to end hosts. Traffic originating from a compromised node can be modified before any correct node witnesses it.

<sup>2</sup>Confidentiality is not the main concern of these protocols. If confidentiality is desired then it is assumed that end hosts are responsible.

- One way hash functions, *eg.* MD5 [110], SHA-1 [94], UHASH [19].
- Hash chains [71], *eg.* TESLA [106].
- Pseudo random functions (PRF) [76].

However some of these protocols do not address an adversary that can modify packets in the protocol's threat models. For example, WATCHERS and HERZBERG are designed against only malicious packet drops, so they require neither cryptographic functions nor key distribution.

A mechanism for key distribution is necessary in order to use these cryptographic functions. Key distribution can rely on either public or secret key infrastructures<sup>3</sup>. Finally, it is assumed that either the administrative ability to assign and distribute shared keys or a public key infrastructure, such as Internet Key Exchange(IKE) [44], or a secure key exchange method, such as Diffie-Hellman [30], is available.

### 2.1.6 Routing

These detection protocols secure forwarding functionality on the data plane. Yet, all of the protocols need more or less a global view of the topology. Network connectivity and routes can be discovered via routing protocols. Secure routing protocols, such as those in Section 1.1.1, should provide secure routing on the control plane.

We can categorize the detection protocols according to the routing protocols that they rely on:

- Static routing<sup>4</sup>: HERZBERG, `OptimisticProtocol`.

These protocols present methods to detect malicious behavior on a single path. However they can be extended to the case of executing a round of proposed detection schemes for every source and destination pair in the network.

---

<sup>3</sup>Digital signatures with public key infrastructure or message authentication codes with pair-wise secret key infrastructure.

<sup>4</sup>In static routing, an administrator manually sets up routing configuration at each node.

- Source routing<sup>5</sup>:
  - Intradomain network: PERLMAN, HERZBERG, HSER.
  - Interdomain network<sup>6</sup>: ACL, PacketObituaries.
  - Both: StealthProbing.

In these protocols, only the source router detects a failure. Announcing this detection to others does not help much, since a correct router receiving this announcement can not trust the source. The detection is utilized only at the source: As a response, the source is responsible for excluding the suspicious link from its routing fabric.

One concern about source routing is that since the route information is embedded into the packets, it increases the processing overhead and packet size, which may possibly cause fragmentation. This is a significant problem for these detection protocols, since the pre-computed fingerprints at the upstream routers are no longer valid<sup>7</sup>. This is discussed further in Section 7.4.4.

- Link state routing<sup>8</sup>: WATCHERS, Protocol  $\Pi_2$ , Protocol  $\Pi_{k+2}$ , Protocol  $\chi$ .

Link state routing protocols have the advantage of adjusting the topology automatically. Once the detection is disseminated to the correct routers, it can be excluded from the routing fabric easily.

- Not specified: SATS, SecTrace.

SecTrace does not specify a routing scheme explicitly. A source router discovers the path to a destination in hop-by-hop fashion similar to Traceroute.

SATS does not specify a routing scheme explicitly, neither. The centralized back-end engine has the global view of the topology and assigns to each router a specific

---

<sup>5</sup>In source routing, each node has a global view of the network topology and computes routes to each destination. The selected route is embedded into the packets that the intermediate routers forward.

<sup>6</sup> For example, BGP, which is a distance vector routing, with multipath support [138, 119, 6].

<sup>7</sup>Unless the fragments are reassembled. Yet, reassembling is unpractical.

<sup>8</sup>For example, OSPF or IS-IS.

subset of traffic to monitor. As it is described, it does not handle mis-routing attacks. However, the related documentation mentions that in order to handle such attacks, link state routing is required.

## 2.2 Threat Model

### 2.2.1 Adversarial Capability

A compromised router can alter traffic and can also behave arbitrarily with respect to the proposed protocols by not participating in the protocol, announcing incorrect reports, or colluding with other compromised routers to launch organized attacks. We use the term *traffic faulty* to indicate a router that alters traffic and the term *protocol faulty* to indicate a router that misbehaves with respect to the proposed protocol. A *faulty* router is one that is traffic faulty, protocol faulty or both. Distinguishing between protocol faulty and traffic faulty behavior is useful, because while it is important to detect routers that are traffic faulty, it is not as critical to detect routers that are only protocol faulty: routers that are only protocol faulty are not altering the traffic flow.

**Attacks on the network data plane:** A compromised router can arbitrarily alter its own forwarding behavior. For example, such a router can drop or modify selected (or all) packets, or divert them to other routers.

We divide up arbitrary behavior into five different threats. These threats completely cover the set of bad behaviors, with respect to data forwarding, which a router can exhibit. When all of these metrics are zero, then no router is forwarding traffic in a faulty manner.

- *Packet loss.* A compromised router can drop any subset of packets. As per Almes et al. [2], loss can be measured as the amount of data arriving at the sink of path-segment subtracted from the amount of data sent from its source.
- *Packet fabrication.* A compromised router can generate packets and inject them into the traffic stream. This can be measured as the number packets which are

reported at the sink of a packet segment but not monitored as being sent by its source.<sup>9</sup>

Misrouting packets can be considered an instance of both packet loss and packet fabrication.

- *Packet modification.* One can consider this threat as a combination of packet loss and fabrication, but it may not be detectable by simply comparing the number of packets arriving at the sink with the number sent from the source. Instead, some summary of the content needs to be maintained, and one measures the number of modified packets.
- *Packet reordering.* A compromised router can reorder packets. Doing this can lead to performance problems or, in the extreme, denial of service. Reordering of packets can effect TCP performance tremendously [16, 18]. There are many reasonable and incompatible methods of measuring the amount of reordering, e.g. [16, 18, 88, 107]. For example in [107]: Given a transmitted stream  $S$  and a received stream  $F$ , remove from both all lost, fabricated and modified packets. Then, find the longest common subsequence  $\ell$  between these modified streams. The amount of reordering is defined as  $|S| - |\ell|$ .
- *Time behavior.* A compromised router can delay traffic or introduce *jitter* to multimedia traffic. Like reordering, doing this can lead to performance problems or, in the extreme, denial of service. There are simple metrics one can use, such as the first  $n$  moments of the inter-packet delay distribution. However, such metrics are notoriously sensitive in packet networks.

**Addressed attacks:** WATCHERS and HERZBERG are effectively designed to detect only packet losses as a result of a Byzantine attack. Consequently, the state requirement for these protocols is minimal. To be robust against alteration, the routers need to

---

<sup>9</sup>None of the Byzantine detection protocols explicitly addressed denial-of-service(DoS) attack. However, standard rate-limit schemes can be applied against DoS attacks.

keep an identity of the packets, namely a fingerprint. PERLMAN, HSER, SecTrace, Protocol  $\Pi_2$  and Protocol  $\Pi_{k+2}$  store this information as well.

It has been shown that reordering of packets can significantly degrade TCP performance [16, 18]. None of the above protocols has taken reordering into consideration as a Byzantine attack. In the framework of underlying traffic validation, Protocol  $\Pi_2$  and Protocol  $\Pi_{k+2}$  address this attack by recording the order between the identities of packets. SecTrace can be extended to detect this kind of attack. However, it is impossible for PERLMAN, HERZBERG, and HSER to handle packet reordering, since they monitor a single packet at each round. And it is impossible for WATCHERS to do so, since it does not keep the identity of packets.

### 2.2.2 Number of Faulty Routers

Attackers can compromise one or more routers in a network.

One cannot depend on faulty servers to detect faulty servers. Compromised routers can cooperate to hide the evidence that a router is faulty. Failure detection can be influenced by the maximum number of adjacent faulty routers as well as by the total number of faulty routers.

Some protocols, for example, PERLMAN, impose an upper bound on the number of faulty routers: If *TotalFault*( $f$ ) holds, then there can be at most  $f$  faulty routers out of  $n$  total number of routers.

Protocol  $\Pi_{k+2}$ , Protocol  $\Pi_2$  and Protocol  $\chi$  impose an upper bound *AdjacentFault*( $k$ ) on the number of adjacent faulty routers. For example, if *AdjacentFault*(3) holds, then there can be no more than 3 adjacent faulty routers in any path.

WATCHERS requires the *good neighbor* and *majority good* conditions. Good neighbor condition states that each router is a neighbor to at least one good router. This is required for each router to be validated by at least one correct router. The majority good condition states that a majority of the routers is good. This is required in order to prevent faulty routers from triggering a new round of the protocol.

HERZBERG, HSER, SATS, and StealthProbing do not require any further restriction other than those in Section 2.1.

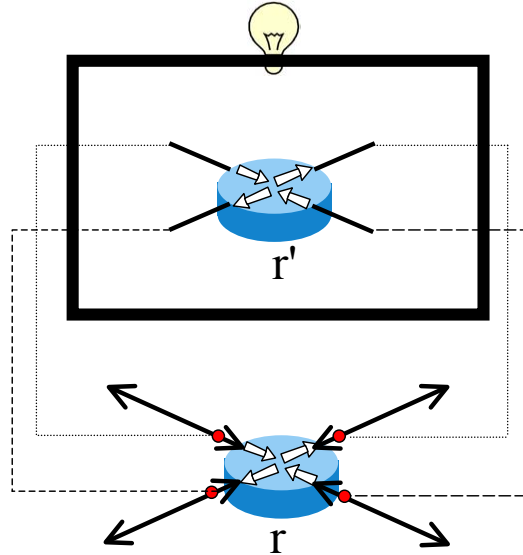


Figure 2.1: Failure detector via active replica / state machine approach.

### 2.3 Centralized Failure Detector via Active Replication

The behavior of a router is deterministic: traffic enters a router and is forwarded on to the next hop towards its destination. Because it is deterministic, the behavior of a router can be verified by a failure detector via an identical replica of that router.<sup>10</sup> For example, in Figure 2.1, a failure detector is implemented with an identical replica  $r'$  of the router  $r$ . In this scheme, the failure detector listens the router  $r$ 's traffic in promiscuous mode and ensures that the replica  $r'$  receives the same input traffic as the router  $r$ . Then the failure detector compares the output traffics of the router  $r$  and the replica  $r'$ . If there is a discrepancy, then a failure is detected and an *alarm* is raised. In this case, either the monitored router is faulty or the failure detector is faulty.

This is an ideal failure detector that detects malicious behavior of compromised routers. However, this scheme has limitations:

<sup>10</sup>This scheme is also called master-checker, active replication, or state machine approach in the literature.

**Complexity of implementation:** First of all, a failure detector must implement the necessary precautions to avoid nondeterminism, such as in scheduling and internal multiplexing. For example, upon receiving routing updates, each router updates its routing tables. If the router and the replica do not update their routing tables simultaneously, then for a short time interval their output traffic might include discrepancies. Another issue is the randomization used in active queue management schemes. Both the router and the replica rely on the same randomization source to generate the same output.

Researchers addressed this issue by implementing a light-weight version of such a failure detector via *traffic validation*: Instead of validating the exact traffic that transits a router, various characteristics of the traffic entering into and leaving various parts of the network can be used for validation. This is discussed in detail in Section 2.4.1.

**Resource requirement:** Furthermore, to implement such a failure detector, as in Figure 2.1, requires additional hardware resources — the identical replica of the router — which might be prohibitively expensive.

This limitation is addressed by implementing such a failure detector in a distributed manner. This requires the participation of uncompromised routers. We discuss this issue in Section 2.4.2.

## 2.4 Distributed Failure Detector via Traffic Validation

The goal is to implement such a detection mechanism as in Figure 2.1 in a distributed manner in the network. Given the system model in Section 2.1, the problem of detecting a compromised router falls to its neighbors: A compromised router can potentially be identified by correct routers when it deviates from exhibiting expected behavior.

As it is mentioned in Section 1, the overall approach can be broken into three distinct subproblems: traffic validation, distributed detection, and response. Next, we study each of these problems in turn. For different threats, we describe a range of appro-

priate and efficient traffic validation functions. We then examine how these functions can be used to build an anomalous behavior detector for compromised routers.

### 2.4.1 Traffic Validation

Traffic validation is the basis for detecting anomalous behavior that determines whether traffic is altered en route. For traffic entering a region of the network, and knowing the expected behavior of the routers in the network, anomalous behavior is detected when the monitored traffic leaving that part of the network differs significantly from what is expected. Traffic validation can be defined in terms of *conservation of traffic*:

**Conservation of traffic:** Some *property* of the *traffic* entering into a *region of a network* must be consistent with the same property of the traffic leaving that part of a network.

There are three design decisions that must be addressed to implement such a mechanism:

- *Traffic to monitor:* What traffic is to be monitored?
- *Conservation of traffic policies:* Which property of the traffic is to be validated?
- *Traffic validation architectures:* What region of a network is to be monitored?

A failure detector based on traffic validation can be as effective as one based on active replication, and the overhead is reasonable. In practice, designing a traffic validation mechanism includes tradeoffs for each design decision above. Hence, implementing a traffic validation mechanism is an engineering problem. In addition, real networks occasionally lose packets due to congestion. Traffic validation needs to accommodate these congestive packet losses. Implementing such validation practically can be quite tricky and requires tradeoffs between the overhead of monitoring, communication and accuracy. We now study the design decisions to implement a traffic validation mechanism in detail.

## Traffic to Monitor

The protocols can be categorized in various ways. Some protocols monitor individual packets while others monitor aggregate traffic. Some protocols are based on *active probing*: they send probe packets periodically; while others deploy a passive approach that simply monitors existing traffic.

**Active probing vs. passive monitoring:** Active probing increases the traffic load in the network, so almost all protocols are based on passive monitoring approach, where the routers passively monitor existing traffic in the network. Only `SecTrace` is originally based on active probing, monitoring the probe packets between pair of routers. It was later replaced with a passive monitoring approach.

**Single packet vs. aggregate traffic:** One of the design decisions made by these protocols is whether to monitor a single packet or aggregate traffic.

`PERLMAN`, `HERZBERG`, and `HSER` monitor a single packet at each round. This leads to two negative outcomes. First, there are thousands of thousands packets that must be forwarded in the network, and these protocols necessitate maintenance of some state for the packet being monitored, such as a timeout clock, reserved buffer, and so on. Thus, if all the packets in the network are monitored, then there is a huge explosion in the state size required at each router. As a result, it becomes impractical to provide Byzantine Robustness for each packet. Another serious limitation is the unfortunate fact that it is quite challenging to attribute a missing packet to a malicious action because normal network congestion can produce the same effect. Modern networks routinely drop packets when the load temporarily exceeds a router's buffering capacity. A packet might be delayed or dropped due to congestion. These are all abnormal but non-malicious behaviors. If the monitored packet happens to be dropped due to congestion, then as a result some routers may be incorrectly identified as faulty. Byzantine Robustness protocols have to be tolerant to these kind of non-malicious abnormal behaviors to some extent.

In contrast, WATCHERS, StealthProbing, SecTrace, Protocol  $\Pi_{k+2}$ , Protocol  $\Pi_2$ , and Protocol  $\chi$  validate aggregate traffic over some period of time. While these protocols compute some state locally for each packet, limiting distributed reconciliation to traffic aggregates amortizes the communication and synchronization overhead (otherwise prohibitive) across many packets. This also makes it feasible to apply a threshold mechanism to distinguish between acceptable bad behavior (e.g. small amounts of packet loss and reordering) and malicious behavior.<sup>11</sup>

As a result, these protocols can tolerate non-malicious abnormal behaviors and can shrink the state size maintained at individual routers. Nevertheless, monitoring all packets in the aggregate traffic might be still too costly. If there are insufficient computational resources, one can easily tradeoff accuracy for overhead by subsampling the packets to be considered. As in Duffield and Grossglauser’s Trajectory Sampling [31], if the same random hash function is used to subsample packets at each end of a path-segment, then each router should observe the same subset of packets. Further, each pair of routers is free to select such sampling functions independently and need not rely on a global secret. SATS is designed with this motivation. For SecTrace and Protocol  $\Pi_{k+2}$ , routers executing the protocols can agree on a subsampling pattern to select a subset of packets to monitor. Meanwhile, subsampling is not applicable in Protocol  $\Pi_2$ .<sup>12</sup>

## Conservation of Traffic Policies

Upon receiving a packet, a router references its routing table to determine the next hop toward the destination, and then forwards the packet. Thus, ideally, the traffic entering a router is equal to the traffic leaving that router. Of course, there is queuing

---

<sup>11</sup>All detection protocols, except Protocol  $\chi$ , have tried to distinguish malicious packet losses from congestive losses using a user-defined threshold: too many dropped packets implies malicious intent. However this heuristic is fundamentally unsound; setting this threshold is, at best, an art and will necessarily create unnecessary false positives or mask highly-focused attacks.

<sup>12</sup>In Protocol  $\Pi_2$ , the sampling pattern is known to every router along the path-segment. Thus, compromised router may attack only the unmonitored packets without being detected.

and processing delay, and packets can be lost due to congestion.

The most precise description of traffic is itself: the exact content of the packets. However, the storage requirements to buffer all packets (as well as the bandwidth consumed by resending them in order to implement distributed detection, discussed later) make this approach impractical. Instead, one can choose less precise properties of the traffic to validate traffic. Many characteristics of the traffic can be summarized far more concisely and can be used to validate different properties of the traffic. In particular, if we concentrate on particular threats – ways in which a malicious entity might alter the traffic – we can limit our effort to detecting just those actions. Some properties are:

1. Conservation of *flow* validates the volume of the traffic, thereby addressing the malicious behavior of dropping packets.
2. Conservation of *content* validates the content of the traffic, thereby addressing the malicious behavior of modifying packets.
3. Conservation of *order* validates the order among the packets that constitute the traffic, thereby addressing the malicious behavior of reordering packets.
4. Conservation of *timeliness* validates the time behavior of the forwarding process, thereby addressing the malicious behavior of delaying packets.

**Conservation of flow:** To address malicious packet loss, the volume of the traffic is the property that is expected to be preserved. This resembles the “conservation of flow” approach used by the WATCHERS, which implements the mechanism, where each router counts the number of packets that it has observed as it monitors traffic over some pre-agreed time interval. Traffic validation is done by comparing the values of these counters. In general, this is a fragile summary function because it only detects actions that cause packet losses, and it assumes that malicious routers cannot fabricate packets to “fudge” the counts appropriately. However, it is extremely cheap to implement, both in the per-packet cost and in the associated overhead to communicate traffic information among routers. It might be possible to extract such information from existing

traffic analysis tools, such as Cisco's Netflow [28], without requiring router modifications though we have not attempted this (there are a number of complexities in how Netflow manages flow records that pose non-trivial synchronization challenges).

WATCHERS and HERZBERG rely on this policy and effectively detect only packet losses as result of an attack launched by compromised routers. Consequently, the amount of state requirement is minimal.

**Conservation of content:** This policy requires the content of traffic to be preserved. To detect modification of packets, a fingerprint,<sup>13</sup> (that is, a one-way hash value), of the payload, in place of a simple counter, can be used. Analogous to conservation of flow, each router then periodically communicates a set of packet fingerprints with other routers for traffic validation, which is calculated via set difference. In addition to detecting packet modification, this approach also detects packet loss, packet fabrication, and misrouting.

One downside to this approach is that it requires storing and communicating a fingerprint for each packet forwarded by a router. This is a significant overhead. One can save significant space and bandwidth by using more sophisticated algorithms for calculating set differences. The simplest approach is to simply use Bloom filters [20] to represent the set of fingerprints. One can then use the population of the bitwise difference between the filters to calculate the size of the set difference. This approach is far cheaper to implement, but comes at some expense in accuracy. More problematic is that it is difficult to know in advance the appropriate parameters for the Bloom filter. A too-small filter can result in significant errors in estimation. A more promising approach is to leverage distributed set reconciliation algorithms [84]. This approach has greater computation overhead than Bloom filters, but it is optimal in bandwidth utilization [84]. In Appendix A, this algorithm is explained in detail.

Packet loss, fabrication, and modification are the attacks widely addressed by the Byzantine detection protocols based on this policy. As they are specified, PERLMAN,

---

<sup>13</sup> The difficulty of computing a fingerprint with changing fields in IP header, such as TTL and checksum, is discussed in Section 7.4.2.

SecTrace, OptimisticProtocol, HSER rely on conservation of content.

**Conservation of order:** This policy further checks the order among the content of traffic. One mechanism for detecting packet reordering is to maintain ordered lists of packet fingerprints rather than simple sets to detect packet reordering. As with conservation of content, this can result in a significant storage overhead. This has a higher overhead than simply computing the size of the set difference.

Only Protocol  $\Pi_2$  and Protocol  $\Pi_{k+2}$  explicitly addressed reordering attacks. However, it would not be hard to extend the protocols monitoring aggregate traffic, such as SecTrace, PacketObituaries to include this kind of attack, but it can not be done with the protocols of PERLMAN, HERZBERG and HSER since they monitor a single packet at each round. And detecting this sort of attack is impossible for WATCHERS since it does not keep the identity of packets.

**Conservation of timeliness:** Conservation of *timeliness* validates the time behavior of the forwarding process in order to address the malicious behavior of delaying packets. Faulty time behavior can be detected by maintaining ordered list of packet fingerprints associated with timestamps. Traffic validation can then be done by computing how much time is spent at each node for a given packet.

None of these protocols, except Protocol  $\chi$ , addresses the faulty time behavior in a systematic way. However, in general, a protocol can be extended to any of these conservation of traffic policies by implementing an appropriate traffic validation mechanism.

## Traffic Validation Architectures

Various existing protocols apply conservation of traffic to different parts of a network, such as per router, per interface, and per path-segment.<sup>14</sup>

<sup>14</sup>A *path-segment* is formally defined in Section 2.4.2, as a sequence of consecutive routers that is a subsequence of a path.

**Per router traffic validation:** WATCHERS first proposed to use a *conservation of flow* (CoF) principle to detect faulty routers. Basically, CoF states that each input to a router should either be absorbed at that router or passed along to another routers. WATCHERS implements per router traffic validation by checking the conservation of flow policy.

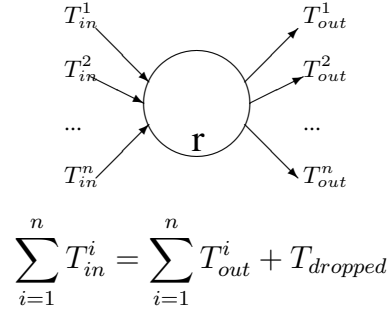


Figure 2.2: Per router conservation of traffic.

In Figure 2.2:

- $T_{in}^i$ , incoming traffic from neighbor  $i$ .
- $T_{out}^i$ , outgoing traffic to neighbor  $i$ .
- $T_{dropped}$ , traffic dropped at router  $r$ .

If  $T_{dropped}$  exceeds some threshold, then router  $r$  is considered to be faulty.<sup>15</sup>

**Per interface traffic validation:** ZHANG and Protocol  $\chi$  implement per interface traffic validation based on conservation of traffic for each output interface<sup>16</sup>.

In Figure 2.3:

- $T_{in}^i$ : incoming traffic from neighbor  $i$  to be forwarded to  $r'$  through  $Q$ .

<sup>15</sup> Traffic originated at router  $r$ ,  $T_{originated}$ , and traffic consumed by router  $r$ ,  $T_{consumed}$ , which we omit for the sake of simplicity, should also be considered:

$$\sum_{i=1}^n T_{in}^i + T_{originated} = \sum_{i=1}^n T_{out}^i + T_{consumed} + T_{dropped}$$

<sup>16</sup> These protocols can be easily extended to a per input interface.

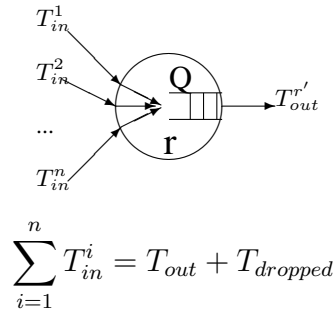


Figure 2.3: Per interface conservation of traffic.

- $T_{out}$ : outgoing traffic to router  $r'$ .
- $T_{dropped}$ : traffic dropped at router  $r$ .

For example, in Protocol  $\chi$ , the behavior of  $Q$  is simulated at neighbor router  $r'$  based the traffic information,  $T_{in}^*$ , it collects from the other neighbors and its traffic information,  $T_{out}^{r'}$ . Protocol  $\chi$  can detect each congestive packet loss individually. Hence, the other packet losses can be attributed to malicious attacks.

**Per path-segment traffic validation:** Most widely deployed traffic validation architecture operate by validating conservation of traffic per *path-segment*. There are two approaches: (1) Only the ends of the path-segment validate the traffic. (2) All nodes along the path-segment participate in validation.

**Per path-segment ends traffic validation:** In Figure 2.4, only end routers, 1 and 6, collect traffic information about the traffic traversing the path  $\langle 1, 2, \dots, 6 \rangle$ . By exchanging this information, the routers can validate conservation of traffic for the path. This mechanism results in *m-accurate* detection, where  $m$  is the length of the path.

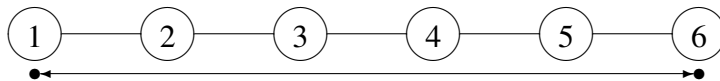


Figure 2.4: Conservation of traffic of a path-segment, end-to-end.

Even though this mechanism does not detect faults accurately, it is widely deployed since it allows the nodes to agree on a sampling pattern and monitor the sampled traffic. This may significantly reduce the cost.

SecTrace, PERLMAN, StealthProbing, SATS, ACL, GOLDBERG, and Protocol  $\Pi_{k+2}$  apply this approach.

**Per path-segment nodes traffic validation:** In Figure 2.5, every router along the path  $\langle 1, 2, \dots, 6 \rangle$  collects traffic information about the traffic traversing. By exchanging this information, the routers can validate conservation of traffic hop-by-hop along the path. This mechanism results in *2-accurate* detection.

HSER, HERZBERG, AWERBUCH, and Protocol  $\Pi_2$  apply this approach.

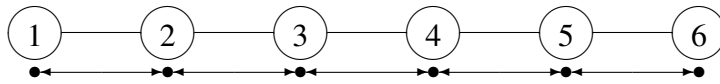


Figure 2.5: Conservation of traffic of a path-segment, along the path-segment.

## 2.4.2 Distributed Detection

Instead of a centralized failure detector, as in Figure 2.1, our goal is to implement such a failure detector distributed in the network using the existing hardware resources: requiring the participation of uncompromised routers.

A compromised router can make arbitrary alterations to its own forwarding behavior. However, given the distributed nature of packet forwarding, it is not possible in general for an adversary to perfectly conceal such behavior. As long as the packets traverse some uncompromised router, there is enough data redundancy to detect the alteration. The detection of a compromised router requires synchronizing the collection of traffic information and distributing the results for detection purposes.

As the failure detector via active replica, in Figure 2.1, has some uncertainty — in case of a detection, it may be either the monitored router or failure detector which is faulty — there will also be some uncertainty in distributed detection of which router

is faulty, since routers collect the information upon which traffic validation is based. For example, suppose router  $r_1$  collects traffic information about packets that traverse  $r_1$ , then a neighboring router  $r_2$ , and then a third router  $r_3$ . Based on the information that  $r_3$  has about the traffic it has seen and the traffic information  $r_1$  has provided,  $r_3$  can determine that packets have been dropped. But,  $r_3$  can't determine whether  $r_1$  is lying about what it claims to have forwarded to  $r_2$  or whether  $r_2$  has dropped the packets. Hence, there is an inherent lack of precision in determining which routers are compromised.

A failure detector reports a suspicion as a *path-segment*, which is defined as a sequence of consecutive routers that is a subsequence of a path.<sup>17</sup> More specifically, a failure detector reports a path-segment if it suspects that a router in that path-segment is behaving in a faulty manner.

In Section 4.2.2, we present a formal derivation of the specification. In short, we cast the problem as a failure detector with *completeness*, *accuracy* and *precision* properties.

**Completeness:** Whenever a router forwards traffic in a faulty manner,

- if *all correct routers* eventually suspect a path-segment containing a faulty router, then a failure detector is *strong-complete*.
- if *at least one correct router* eventually suspects a path-segment containing a faulty router, then a failure detector is *weak-complete*.

**Accuracy:** A failure detector is *accurate* if, whenever a correct router suspects a path-segment, then there is at least one faulty router in that path-segment.

**Precision:** A failure detector also has a *precision*, which is the maximum length of a path-segment it suspects.

A failure detector must be complete and accurate, and preferably with a *high precision*. Implementing such distributed detection involves tradeoffs among precision,

---

<sup>17</sup>For example, if a network consists of the single path  $\langle r_1, r_2, r_3, r_4 \rangle$  then  $\langle r_2, r_3 \rangle$  is a path-segment, but  $\langle r_1, r_3 \rangle$  is not because  $r_1$  and  $r_3$  are not adjacent.

weak/strong-completeness, and the overhead of monitoring and communication. Various detection protocols address these design decisions in different ways and we study such protocols in Chapter 3.

Compared to weak-completeness, strong-completeness is more desirable property since every correct router detects the fault. Given a weak-complete detector, a strong-complete detector can be implemented but the implementation may not be simple and some precision would be lost. For example, consider a source router  $r_s$  that detects a link  $\langle r_1, r_2 \rangle$  as faulty. Announcing this detection, the other correct routers in the network have to consider the case that  $r_s$  is faulty as well. On the other hand, in some cases, having a weak-complete detector is sufficient in order to take proper response: for example, relying on source routing, the router  $r_s$  may only update its own routing table excluding the suspected  $\langle r_1, r_2 \rangle$ .

### Properties of Protocols

In terms of our specification, WATCHERS is *accurate* with a *precision* of 2. However, it is not complete since it has a flaw as explained in Section 3.1. The flaw can also be fixed as suggested in Section 3.1, in which case the improved protocol is strong-complete.

The protocols that are based on per path-segment nodes traffic validation, including HSER, HERZBERG, and AWERBUCH, are weak-complete and accurate with precision 2.

The protocols that are based on per path-segment ends traffic validation, including SecTrace, PERLMAN, StealthProbing, SATS, ACL, and GOLDBERG, are weak-complete and accurate with precision  $M$ , where  $M$  is the length of the path-segment that is monitored. For example,  $M$  is the length of the monitored path between the source and destination for PERLMAN, StealthProbing, ACL. In SATS, the centralized backend system decides which path-segments to be monitored.

Most of the protocols are originally presented as weak-complete, such that only the source router detects the failure. However, they can easily be extended to

strong-completeness by requiring that the source announce the signed traffic information intended to lead to the detection.

ZHANG, which is based on per interface traffic validation, is strong-complete, accurate with precision 2.

### 2.4.3 Response

Once a path-segment  $\pi$  is detected as containing compromised routers, some countermeasure should be taken. Of course, the most important countermeasure is to log the suspicion and alert the administrator of the affected routers. Less obvious is how routers should react to a detection in the short term.

Suppose that some path-segment  $\pi$  is detected. An obvious countermeasure would be to remove all of the routers in  $\pi$  from the routing fabric. By doing so, we avoid using any router that has been suspected of being compromised. Doing this, though, could also have a serious, and perhaps unnecessarily high, impact on network performance. A less aggressive countermeasure would be to only remove the path-segment  $\pi$  from the routing fabric: routers update their forwarding tables such that no traffic traverses along the suspected path-segment  $\pi$  anymore. In doing so, we may allow compromised routers to keep forwarding packets, but only along paths over which no faulty behavior has been observed.

We have chosen the second approach because of its less disruptive behavior. If only a single interface is compromised (today's interfaces are effectively their own CPUs) then only the path-segments incident on that interface will be excluded. If a router is disrupting traffic along several paths, then each of these paths will be separately detected and then routed around. Finally, if a router is uniformly malicious (i.e., causes traffic validation to fail for all traffic passing through it) then all intersecting path-segments will be excluded and the router will be completely isolated.

## **Acknowledgement**

Parts of Chapter 2 are reprints of the material as it appears in UCSD Technical Report, CS2007-0899, 2007, by Alper Tugay Mizrak, Keith Marzullo and Stefan Savage.

## Chapter 3

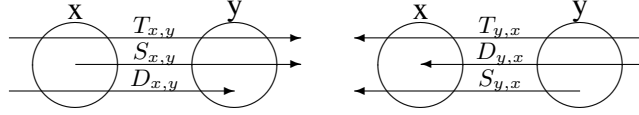
# Literature Review

This chapter presents a literature review of the existing failure detection protocols. We study various protocols proposed as a countermeasure for the attacks on the network data plane. First, we present each traffic validator implemented by the protocols as a single centralized service, which is explained in Section 2.3. Next, we examine the various design decisions for each existing protocol and study how the protocols have implemented such failure detectors by distributing them in the network. As such, this chapter presents the state of art in these failure detection protocols today.

### 3.1 WATCHERS: A distributed network monitoring approach

The WATCHERS protocol, which detects and isolates faulty routers based on a distributed network monitoring approach, was developed (and criticized) at the University of California, Davis from 1997 through 2000 [27, 21, 56]. A faulty router is defined as one that drops or misroutes packets, or that behaves in an arbitrary manner with respect to the WATCHERS protocol. Cheung and Levitt [27] first proposed to use a *conservation of flow principle* (CoFP) to detect faulty routers. Basically, CoFP states that each input to a router should either be absorbed at that router or passed along to another routers.

As shown in Figure 3.1, each router counts the number of bytes it has received and forwarded through each link during an agreed-upon time interval. Each router then



- $T_{x,y}$ : for transit packets that pass through both  $x$  and  $y$ .
- $S_{x,y}$ : for packets with source  $x$  that pass through  $y$ .
- $D_{x,y}$ : for packets with destination  $y$  that pass through  $x$ .

Figure 3.1: WATCHERS: Transit packet byte counters

floods snapshots of its counters. Once a router has these counters, it uses a two-phase protocol to detect faulty routers. The two phases are:

1. **Validation:** A router  $a$  compares, for each neighbor  $b$ , its counters – for the  $\langle a, b \rangle$  link – with those of  $b$ . If the counters do not agree, it detects its neighbor  $b$  as faulty. Similarly, for each neighbor  $b$  and each of its neighbor  $c$ ,  $a$  compares the  $\langle b, c \rangle$  link counters of  $b$  with those of  $c$ . If these counters do not agree, then  $a$  knows that at least one of  $b$  and  $c$  is faulty, and so  $a$  does nothing further with  $b$ ; it assumes that  $b$  will detect  $c$  as faulty or vice versa.
2. **Conservation of flow test:** If the validation phase is passed successfully, then  $a$  checks if each neighbor  $b$  preserves CoFP. It does so by computing the incoming transit flow  $I_b$  and the outgoing transit flow  $O_b$  of router  $b$ :

$$I_b = \sum_{\forall c|b \leftrightarrow c} (S_{c,b} + T_{c,b}) \quad O_b = \sum_{\forall c|b \leftrightarrow c} (D_{b,c} + T_{b,c})$$

If  $|I_b - O_b| > T$  for some threshold  $T$  then  $a$  diagnoses  $b$  as *faulty*.

In this scheme, each router maintains six counters for each of its neighbors.<sup>1</sup> Thus, if  $R$  is the maximum connectivity in the network, then the space cost per router of this protocol  $O(R)$ . Since all counters are compared over the same time interval, all of the routers periodically synchronize with each other.

<sup>1</sup>In fact, each router maintains seven counters for each of its neighbors. The seventh counter tracks packets misrouted by that neighbor. Whenever this counter is nonzero, the associated router is identified as faulty.

Later, the architects of WATCHERS noticed that this algorithm was not sufficient to detect *consorting faulty routers* [104], defined as faulty routers launching a coordinated attack and cooperating to hide each other’s malicious behavior. For example, in Figure 3.3, let  $a$  send packets to  $e$  through  $b$ ,  $c$ , and  $d$ . If  $c$  and  $d$  are consorting faulty routers then they can drop all packets and still hide this attack by simply increasing their  $D_{c,d}$  counters rather than  $T_{c,d}$ . With the motivation of this scenario, Bradley *et al.* extended the results in [27] and presented the final version of the WATCHERS protocol [21]. In this version, each router maintains a separate set of counters for each neighbor and final destination of each packet. In the example, when  $a$  sends the packet, it updates its  $S_{a,b}^e$  counter.  $b$  updates its  $T_{b,c}^e$  counter after forwarding the packet.  $c$  and  $d$  now cannot simply drop the packets and hide the attack just by updating some of their counters. In this scheme, the space required at a router is  $O(RN)$ , where  $N$  is the total number of routers in the network.

WATCHERS was designed assuming:

- *Link state condition:* Good routers agree on the exact topology of the network.
- *Good neighbor condition:* Each router is a neighbor to at least one good router.
- *Good path condition:* Each pair of good routers has at least one path of only good routers connecting them.
- *Majority good condition:* A majority of the routers is good. This is required to prevent faulty routers from triggering a new round of the protocol.

WATCHERS is the most similar approximation to the failure detector in Figure 2.1, which detects and isolates faulty routers based on a distributed network monitoring approach. A faulty router is defined as one that drops or misroutes packets, or that behaves in an arbitrary manner with respect to the proposed protocol.

The traffic validator that WATCHERS implements is given in Figure 3.2 as a centralized service. WATCHERS validates the conservation of flow property of the aggregate traffic entering into each router in the network. If the difference between

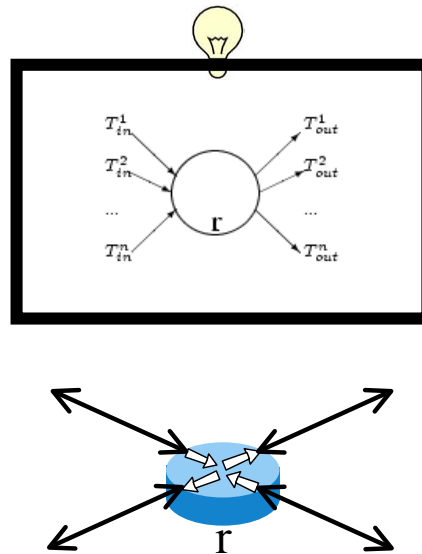


Figure 3.2: Failure detector via a traffic validator per router.

the volumes of the traffic entering into and leaving the router exceeds a user-defined threshold, then a failure is detected and an alarm is raised. This threshold is needed to avoid false positives as a result of congestive packet losses.

WATCHERS implements this failure detector by requiring all of the neighboring routers of a router  $r$  to synchronize with each other, to count how many bytes they have received from and forwarded to  $r$  during an agreed-upon time interval, to distribute the snapshots of their counters to the others by flooding, and finally to validate the conservation of flow property.

If a neighbor router  $r_n$  can not validate the router  $r$ , then  $r_n$  announces that the link  $\langle r_n, r \rangle$  is suspicious and  $\langle r_n, r \rangle$  is removed from the routing fabric.

Two years later, another group at UC Davis (Hughes *et al.* [56]), argued that CoFP is inappropriate to use as a security mechanism in network protocols. They mentioned three general scenarios in which WATCHERS does not work:

- Those for which WATCHERS can be fixed with small modifications in the protocol such as source routing, premature aging.
- Those that are not addressed by WATCHERS, such such as packet modification and

packet fabrication. These could be addressed with a more general traffic validation mechanism.

- Those that represent attacks on the control plane such as ghost routers, and “hot potato” examples in [56] where faulty routers announce incorrect LSPs.

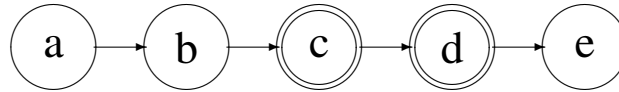


Figure 3.3: WATCHERS: Consorting routers

Perhaps more interestingly, they did not notice that WATCHERS failed to detect one case of consorting routers. Consider two faulty routers  $c$  and  $d$  in Figure 3.3. Assume that there is another (unshown) set of bidirectional links connecting  $a$ ,  $b$  and  $e$  so that the *good path condition* is satisfied. Thus, all of the system requirements are met. Assume that  $c$  drops packets it sends along the  $\langle c, d \rangle$  path but it does not reflect this in  $T_{c,d}^e$ . Router  $d$  can have a correct value of  $T_{c,d}^e$  that is inconsistent with  $c$ 's counter  $T_{c,d}^e$ , which means that their neighbors  $b$  and  $e$  will not perform conservation of flow test for  $c$  or  $d$  respectively. In other words, router  $d$ , being faulty, may not detect  $c$  as faulty.

This flaw can be fixed: a router that detects its neighbors' counters are inconsistent expects those neighbors to detect each other and announce this detection by flooding. For the example given above, routers  $b$  and  $e$  expect to receive a detection of  $\langle c, d \rangle$  within some time interval. Otherwise,  $b$  detects  $\langle b, c \rangle$  and  $e$  detects  $\langle d, e \rangle$ .

In terms of our specification, it is *accurate* with a *precision* of 2. WATCHERS is not complete since it has a flaw as explained above. It can also be fixed as suggested above, in which case the improved protocol is strong-complete.

Our concerns about WATCHERS differs from the criticisms of [56]. Generally speaking, there are two limitations of WATCHERS:

- First, there is no specification of the problem it solves, which makes it hard to compare with other protocols. The main drawback of WATCHERS is its weak traffic validation, which is designed for a restrictive threat model: it addresses

only malicious packet drops and misroutes. Several researchers have subsequently developed protocols with more general traffic validation mechanisms addressing a comprehensive set of attacks.

- Second, consorting faulty routers are the faulty routers launching a coordinated attack and cooperating to hide each others malicious behavior. WATCHERS addressed the issue by requiring each router to maintain a separate state for every neighbor and destination pair in the network. The amount of state each router must maintain is bounded from above only by the total number of routers in the network. Other protocols addressed this problem of consorting faulty routers with a different approach: they validate traffic over path-segments. We discuss this next.

### 3.2 HSER: Highly secure and efficient routing

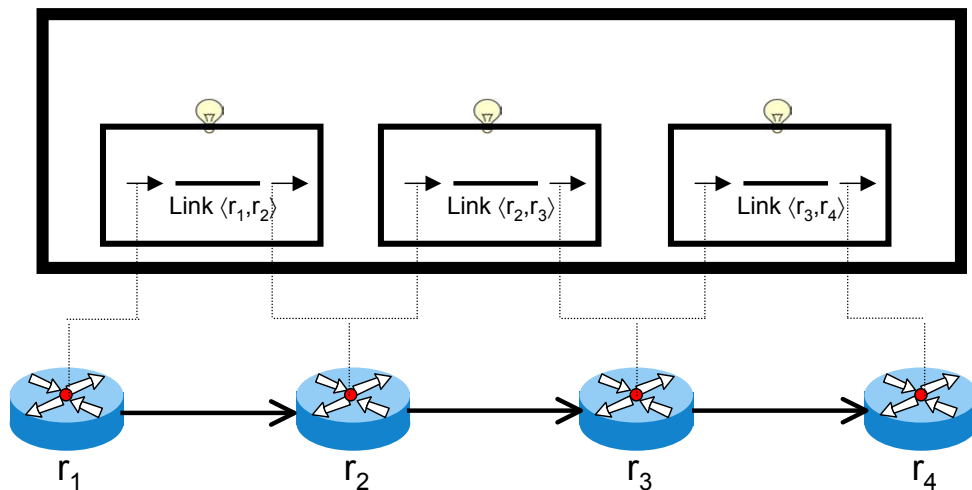


Figure 3.4: Failure detector via a traffic validator per path-segment nodes.

[11] presents Highly Secure and Efficient Routing (HSER), a combination of source routing, hop-by-hop authentication, a-priori reserved buffers<sup>2</sup>, sequence num-

<sup>2</sup>HSER uses a-priori reserved buffers for a pre-defined number of outstanding packets per source. This guarantees

bers, timeouts, end-to-end reliability mechanisms, and fault announcements. As it is noted in [11], while none of these individual mechanisms is novel by itself, it is the combination of them that delivers Byzantine robustness and detection.

The traffic validator that HSER implements is given in Figure 3.4 as a centralized service. HSER validates the conservation of content property of a single packet that is monitored along the path from the source to the destination. If any router along the path discovers that its neighbor has lost or altered the packet, then a failure is detected and an alarm is raised.

HSER implements such a failure detector distributed in the network by requiring each router along the path to compute a fingerprint for the monitored packet, and to keep a timeout, and finally to validate the conservation of content property with its neighbors. Upon receiving a packet, the router first validates the authenticity and forwards the packet to the next hop towards the destination. After forwarding the packet, the router sets a timeout for the worst case round trip time to the destination from itself. If authenticity of the packet is not verified or the timeout expires, then the router generates a *fault announcement*, including its neighbor and itself, to send back to the source.

HSER relies on source routing. As a response, upon receiving a fault announcement, the source router computes a new route to the destination excluding the detected link from its routing fabric.

In terms of our specification, HSER is *weak-complete* – since only the source detects a failure – and *accurate* with a *precision* of 2.

The overhead of this approach is prohibitively high, since for every source and destination pair, all of the routers along the path must participate in the detection. Researchers have developed other protocols based on traffic validation per path-segment ends in order to implement a feasible detector for practical deployment. The tradeoffs are:

- Give up precision by only validating at the end routers of the path-segment, in

---

that packets are never dropped due to congestion.

which case none of the intermediate routers along the path participates in detection.

- The end routers of the path-segment can decide on a sampling pattern and keep track of only the chosen packets. This method may help decrease overhead significantly. However, attacks to unmonitored packets would not lead to a detection, and the accuracy in detection would be decreased.

HSER is based on traffic validation per path-segment nodes. Other protocols based on this approach are HERZBERG, *PacketObituaries*, and AWERBUCH. HSER basically extends  $\text{HERZBERG}_{\text{end-to-end}}$ <sup>3</sup> to real network settings, and works with a more general threat model, considering attacks such as dropping, modifying, delaying packets, etc. Our  $\text{Protocol } \Pi_2$ , in Section 5.1, is also based on traffic validation per path-segment nodes.

### 3.3 HERZBERG: Early detection of message forwarding faults

Herzberg and Kutten [49] present an abstract model and various protocols for Byzantine detection based on timeouts and acknowledgments from the destination and possibly from some of the intermediate nodes to the source.

Their significant contribution to the field is the first formal specification of the problem within a system model designed for only the transmission of a single message along a fixed path consisting of processors. The task is to deliver a message from a source processor to a destination processor or, if there is a fault along the path, to detect the fault location in minimal time with low communication complexity.

The requirement of information from intermediate nodes offers a trade-off between fault detection time and message communication overhead. This trade-off is analyzed within the given abstract model.

The following protocols address this trade-off in various ways:

---

<sup>3</sup> HERZBERG is designed to detect packet drops and for only a fixed path.

- $\text{HERZBERG}_{\text{end-to-end}}$  fault detector: When a destination  $d$  receives a packet, it sends back an ack to the source  $s$  along the same path. Each node  $r$  along the path keeps a timeout clock. If it does not receive an ack or a fault announcement from its neighbor  $r + 1$  in time, it detects  $r + 1$  and announces  $\langle r, r + 1 \rangle$  as faulty. For this protocol, the communication complexity is optimal, since only a single ack is sent per message. However, it suffers from high time complexity to detect the failure.
- $\text{HERZBERG}_{\text{hop-by-hop}}$  fault detector: Each node along the path sends back an ack after forwarding a data packet and keeps a timeout clock for the other intermediate nodes between itself and the destination. In this protocol, the faulty link can be identified in optimal time, but the message complexity is high, since each intermediate node sends an ack back to the source immediately upon receiving the data packet.

This protocol relies on the same idea as  $\text{PERLMAN}_d$ : acks from intermediate nodes. However, the problem of colluding faulty nodes mentioned in Figure 3.8 for  $\text{PERLMAN}_d$  does not constitute a problem for  $\text{HERZBERG}_{\text{hop-by-hop}}$ . This is because all intermediate nodes in  $\text{HERZBERG}_{\text{hop-by-hop}}$  actively participate in detection, unlike  $\text{PERLMAN}_d$ , where only the source node is responsible for detection.

- $\text{HERZBERG}_{\text{optimal}}$ : By selecting some of the intermediate nodes to send an ack to some chosen intermediate nodes, they developed an efficient detection protocol,  $\text{HERZBERG}_{\text{optimal}}$ , in terms of both communication and time complexity.

### 3.4 PacketObituaries: Packet obituaries

Argyrazi *et al.* [7] present `PacketObituaries` for a slightly different setting: Wide area interdomain networking. BGP is the routing protocol used to maintain a table of reachability among autonomous systems (AS) [47].

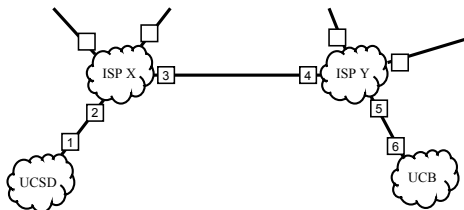


Figure 3.5: PacketObituaries in wide area interdomain network.

PacketObituaries proposes to mount *accountability boxes* (A-boxes) on the external links of each border router, as shown in Figure 3.5. For each packet that an A-box observes, it computes a fingerprint and records the AS-number of the autonomous system where the packet is last seen.

This is the only protocol that is designed for hard-wired networks and yet it does not assume that the global view of the network topology is available to participating nodes. So, A-boxes learn their neighbors by using a discovery process, i.e. by sending *discovery requests* periodically on the links.

The authors discuss two versions of the protocol. The first, `PacketObituarieshop-by-hop`, requires A-boxes to exchange traffic information hop-by-hop periodically. Upon forwarding a packet, an intermediate A-box sets a timeout to the worst case round trip time to the destination. If the A-box receives traffic information from its downstream neighbor, then the A-box replaces the last-AS-number of the corresponding packet fingerprint. If the timeout expires, the last-AS-number is set to the local AS-number. Finally, the A-box sends its traffic information back to the upstream neighbor. Every A-box disseminates the information of last-AS-number back to its upstream neighbor all the way up to the source.

The second version, `PacketObituariessource`, requires each intermediate A-box to send traffic information of a packet fingerprint as well as the local AS-number to the source. Upon collecting all the traffic information, the source is responsible for validating and for determining how far the packet was able to travel towards its destination.

PacketObituaries is primarily designed for a non-malicious setting.

Later the threat model is extended to handle malicious ASes by disseminating traffic information digitally signed. A malicious *downstream* AS that drops a packet and claims not to have received it is indistinguishable from a malicious *upstream* AS that drops the packet and claims to have already forwarded it. So, detection must include two neighboring ASes.

Upon a detection, the source chooses another path excluding the suspicious link as a countermeasure. This assumes that a multipath discovery mechanism for BGP is available, such as Platypus [119], NIRA [138], WRAP [6].

### 3.5 **AWERBUCH: An on-demand secure routing protocol resilient to Byzantine failures**

Awerbuch *et al.* [14] present a failure detection protocol, which we name AWERBUCH in this dissertation, against Byzantine failures including individual or colluding nodes that drop, modify, or mis-route packets. They integrate their adaptive probing technique into an on-demand routing protocol for ad hoc wireless networks.

Similar to SecTrace, where the source searches the failure linearly on the path towards the destination, for AWERBUCH the source performs a binary search on the path. The source specifies a probe list of intermediate nodes. Each node in this list participates in the validation, in addition to the destination node. If the source and destination can not validate traffic between themselves, then during the next round, the source adds the node in the middle into the probe list. This path sub-division process continues until the detected failure corresponds to a link. If the faulty node continues to introduce discrepancy into the monitored traffic, the source will identify a faulty link after  $\log M$  rounds, where  $M$  is the length of the path. The authors deploy a link weight management scheme to avoid faulty links in route discovery process.

In terms of our specification, AWERBUCH is *weak-complete* – since only the source detects a failure – and *accurate* with a *precision* of 2.

### 3.6 SecTrace: Secure Traceroute

Traceroute [57] is a protocol used to determine the route between two nodes in a network. The source simply sends packets to the destination with increasing time-to-live (TTL) values. Upon receipt of a packet with an expired TTL, an intermediate node sends back an *ICMP time exceeded* packet. The source discovers the path to the destination hop-by-hop.

Padmanabhan and Simon [98] developed Secure Traceroute, which securely traces the path of existing traffic. They named the protocol *SecTrace* for short in [78], where they extended their research to use the protocol in the context of community wireless networks.

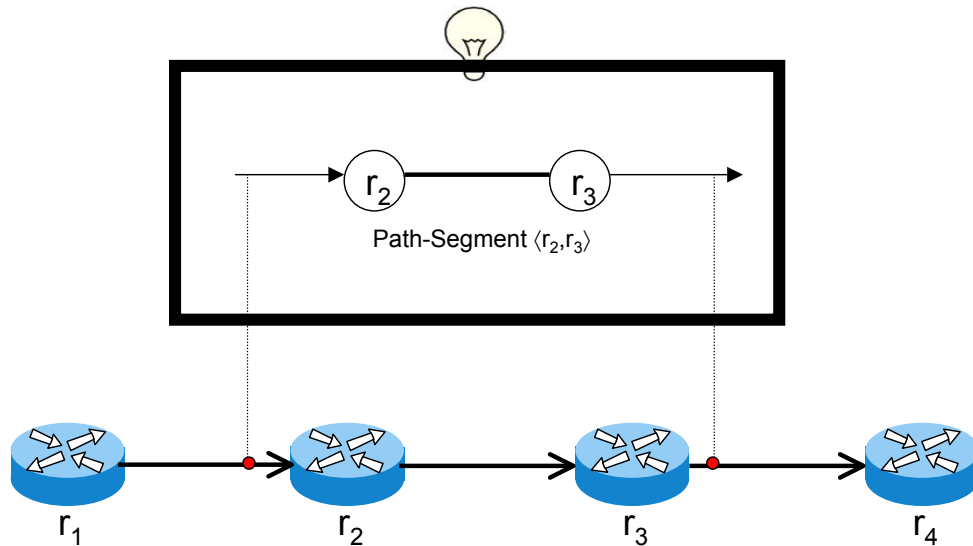


Figure 3.6: Failure detector via a traffic validator per path-segment ends.

*SecTrace* is developed as a practical tool to securely trace the path of existing traffic towards a particular destination from a source. It proceeds hop-by-hop similar to Traceroute: at each round, the source validates the traffic between itself and an intermediate router towards the destination.

The traffic validator that *SecTrace* implements is given in Figure 3.6 as a centralized service. *SecTrace* validates the conservation of content property of the

aggregate or sampled traffic<sup>4</sup> between the source router and an intermediate router. If the source detects that there is discrepancy in the traffic, then a failure is detected and an alarm is raised.

SecTrace implements such a failure detector distributed in the network by requiring only the end routers of the monitored path-segment to synchronize with each other and to compute fingerprints for the traffic between themselves for an agreed-upon time interval. At the end of the round, the corresponding intermediate router sends back the information it has collected and the identity of the next expected router towards the destination. Upon receiving this information, the source router validates the conservation of content property: if the source validates the traffic, then it initiates another SecTrace round with the next intermediate router towards the destination; otherwise, the source detects a failure.

For example, in Figure 3.6, a path-segment of  $\langle r_1, r_2, r_3, r_4 \rangle$  is monitored during the given traffic validation round and only the source  $r_1$  and the corresponding intermediate router  $r_4$  implement the distributed failure detector. If the source  $r_1$  detects that there is discrepancy in the traffic, then a failure is detected and an alarm is raised: (1) Either one of the intermediate routers  $\{r_2, r_3\}$  is traffic faulty introducing discrepancy into the monitored traffic. (2) Or, the failure detector, which is implemented by  $r_1$  and  $r_4$ , is protocol faulty: at least one of  $\{r_1, r_4\}$  is faulty.

In terms of our specification, SecTrace is *weak-complete* – since only the source detects a failure – and *accurate* with a *precision* of  $k$ , where  $k$  is the length of the monitored path-segment.

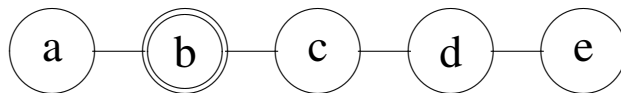


Figure 3.7: SecTrace: Byzantine faulty router.

On the other hand, in [98], the authors require that the source detects the link between the corresponding intermediate router and its upstream neighbor. For example,

<sup>4</sup>It can adopt both active probing and passive monitoring approaches.

in Figure 3.7, if the source  $a$  could not validate the traffic with  $d$ , then it would detect  $\langle c, d \rangle$  as faulty. The reasoning behind this approach is that the source  $a$  was able to validate the same traffic up to the upstream neighbor  $c$  at the previous validation round, so either  $c$  or  $d$  must be faulty introducing discrepancy into the traffic. However, this approach violates the accuracy property. Assume that only the router  $b$  is faulty manipulating the traffic only after the source  $a$  validates the traffic with  $c$ . Consequently,  $\langle c, d \rangle$  would be detected where neither  $c$  nor  $d$  is faulty. In this scenario,  $b$  is able to hide its attack and to frame correct routers by carefully choosing a time to start its attack.

In [98], another scenario is discussed: A malicious router confines its attack to periods of time during which there is no `SecTrace` activity. To address this problem, the authors propose to give occasional indications of `SecTrace` activity, such as by continuously sending round initialization packets pretending to monitor the traffic, while in reality doing nothing.

This strategy makes it harder, but not impossible, for a misbehaving router to decide when to mount an attack. There is still non-zero probability that a faulty router would not be detected, and what is more, that correct routers would be suspected as faulty, as in Figure 3.7.

Only the source router detects a failure as a link. As a response, they propose three different approaches:

1. The source tries to route the traffic around the detected link using source routing.

From the protocol specification, it is apparent that `SecTrace` does not rely on source routing, since it requires the corresponding nodes sending the next expected node to the source router at the end of each round. Unless source routing is used, it is not clear how to realize this countermeasure.

2. The source notifies the downstream routers, expecting them to make the appropriate routing adjustments avoiding the suspected routers.

Consider a downstream router receiving a detection announcement of  $\langle r_1, r_2 \rangle$  from a source  $s$ . The link  $\langle r_1, r_2 \rangle$  can not be excluded from the routing fabric

immediately. It might be the case that  $s$  is faulty announcing bogus detections blaming correct routers. Therefore, only the traffic passed through  $s$  must be rerouted around the link  $\langle r_1, r_2 \rangle$ .

3. The source alerts the administrator of the suspected routers.

Other protocols based on traffic validation per path-segment ends are PERLMAN, StealthProbing, SATS, ACL, and GOLDBERG. Our Protocol  $\Pi_{k+2}$ , in Section 5.2, is also based on this approach.

### 3.7 PERLMAN: Network layer protocols with Byzantine robustness

The earliest work on fault-tolerant forwarding is also due to Perlman [104]. In her PhD thesis, Perlman presented network layer protocols with Byzantine robustness and Byzantine detection. These results are also summarized in her book [105].

In her PhD thesis, Perlman categorized network layer protocols into four robustness levels:

- Simple robustness: Network layer algorithms that are robust against simple failures, such as node or links failures.
- Self-stabilization: Such algorithms do not provide any guarantees in the presence of a malfunctioning node. However, once malfunctioning nodes are disconnected from then network, these algorithms guarantee convergence to the correct behavior.
- Byzantine robustness: An algorithm is defined to be Byzantine Robust if it exhibits correct behavior in the face of arbitrarily malfunctioning (Byzantine failure) nodes. The correct behavior in this context is to be capable of delivering a data packet from source to destination<sup>5</sup>.
- Byzantine detection: In such algorithms, the identity of the faulty components can be discovered in the face of Byzantine failures.

---

<sup>5</sup>It is assumed that there exists a good path between every source and destination.

In this dissertation, our interests are in protocols with Byzantine robustness and detection; as Perlman indicated: “... the ideal network would have both Byzantine detection and Byzantine robustness.”

First, she developed *robust flooding*, which is a method to deliver a packet reliably to all correctly operating routers. This requires the *good path condition*, which states that each pair of nonfaulty routers is connected by at least one path of zero or more nonfaulty routers. Robust flooding was designed to be used for public key distribution and broadcasting *link state packets* (LSP), which is a necessary part of link state protocols.

Next, Perlman developed a data routing protocol with Byzantine robustness. This protocol uses multipath routing designed for *TotalFault*( $f$ ): no more than  $f$  Byzantine faulty nodes exist in the system. The source router computes  $f + 1$  disjoint paths to the destination and forwards the packet over those  $f + 1$  paths. This protocol provides Byzantine robustness but not Byzantine detection.<sup>6</sup>

Perlman also developed a novel method for robust routing on top of a link state protocol with Byzantine detection. In this protocol, the source router first computes a route based on its local database and then sends a digitally signed *route-setup packet* along the chosen route. Each intermediate router on the route verifies the signature and allocates the necessary resources for the data packet to avoid congestion losses. If the source router receives an acknowledgment of route-setup from each intermediate router on the chosen route, then it sends the data packet. The destination router sends back another ack, if the data packet reaches it. If the source does not receive an ack for the data packet from the destination, then it determines that the chosen route is not reliable and computes a new node-disjoint alternative route. In the rest of this dissertation, we name this protocol PERLMAN and compare this protocol with others that were designed for Byzantine detection.

Once PERLMAN detects a path as faulty, it treats all the routers in that path as faulty and avoids sending data through them. In her Ph.D. thesis, Perlman also discussed

---

<sup>6</sup>Byzantine robustness does not imply Byzantine detection. For example, the simplistic algorithm might flood each data packet into the network, which is prohibitively expensive.

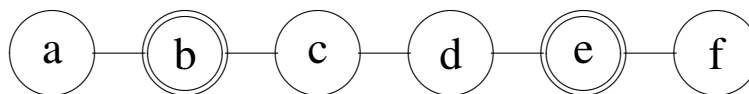


Figure 3.8: PERLMAN: Colluding routers.

another strategy, which we name  $\text{PERLMAN}_d$ , that increases the detection accuracy by means of having every intermediate router send an ACK back to the source, for every packet that it forwards.<sup>7</sup> Perlman concludes that this is not a good strategy, due to high message complexity. However, the main reason that she rejects this scheme is that it is neither *accurate* nor *complete*: A faulty router could fool a correct router into detecting two correct routers as faulty. She gives the following example: In Figure 3.8, assume that routers  $b$  and  $e$  are faulty, and are colluding to hide their attack. Now suppose that  $a$  sends a data packet to  $f$ . If  $e$  fails to forward the data packet to  $f$  and  $b$  fails discriminatorily to forward ACK packet from  $d$ , then  $a$  would receive ACKs from  $b$  and  $c$ , but not from any router further. Subsequently,  $a$  concludes that one of  $c$  or  $d$  must be faulty and detects  $\langle c, d \rangle$ , even though both routers are correct. Thus, the protocol is not accurate. Since  $a$  could not detect the faulty behavior of  $e$  dropping packets, the protocol is not complete either.

### 3.8 StealthProbing: Stealth probing

Avramopoulos and Redfox present `StealthProbing` [12], an *end-router-to-end-router failure detection mechanism*. The protocol creates an *IPsec* [64] tunnel and encrypts the traffic between the corresponding routers using an *Encapsulating Security Payload* [63] module. It can use either active probing or passive monitoring. In the case of active probing, it diverts the probing traffic into the encrypted channel, which ensures that the probe traffic is undistinguishable from the normal data packets. For passive monitoring, the authors propose a sampling method similar to `TrajectorySampling`.

<sup>7</sup>This is the underlying strategy of some of the the protocols proposed later: `SecTrace` and `HERZBERG`.

`StealthProbing` determines the availability of a path between two end routers. This approach only tests for gross connectivity, does not localize the problem.

### 3.9 SATS: Secure split assignment trajectory sampling

Duffield and Grossglauser [31] proposed `TrajectorySampling`. For each packet, every router in the network computes a hash value over the packet content that does not change along the path. If a packet's hash value falls into the predetermined hash range, the packet is sampled. Since identical hash function and hash range are used, every router in the network monitors the same sampled traffic. For each sampled packet, a router computes a fingerprint of the packet and reports it to the centralized measurement system. After collecting these reports, the measurement system can reconstruct the path, which is called trajectory, that the monitored traffic traversed.

Lee *et al.* [75] proposed Secure Split Assignment Trajectory Sampling (SATS) extending the idea of `TrajectorySampling` to detect malicious routers. In SATS, the centralized backend system assigns *different* hash ranges to every pair of routers in the network through encrypted channels, which ensures that the hash range assigned to a router is secret. Every router monitors the traffic falling into its assigned hash ranges and reports back to the backend system, which runs the detection process.

If the backend system detects an inconsistency between routers  $r_i$  and  $r_j$  then all the routers between them, including  $\{r_i, r_j\}$  are suspected to be faulty.

### 3.10 ACL: Availability centric routing

Similar to `PacketObituaries`, Wendlandt *et al.* present *Availability Centric Routing* (ACR) [132], which is also designed for interdomain networking and based on multipath routing. It is assumed that multipaths are provided or an extension of multi-paths to BGP is available [128]. The main idea is that an end host monitors the end-to-end availability to a destination and then switches to another path if the performance of a path is not satisfactory.

### 3.11 GOLDBERG: The role of cryptography in network accountability

[39] presented a formal theoretical framework, in a game setting, for defining the problem of Byzantine detection of faulty nodes in a path with game-based definitions. Their major contribution is the negative result that proves that any *Byzantine detection protocol* on the data plane requires a key infrastructure, cryptographic operation, and dedicated storage at every node.

They presented two end-to-end detection protocols, which use sampling to decrease the processing overhead traffic between a source and destination. The first is called `PepperProbing`, in which the source and destination sample the traffic between them using a cryptographic hash function with pairwise symmetric keys, similar to `TrajectorySampling` and `SATS`. The second protocol is `SaltProbing`, in which public keys are used to sample with a timing strategy, similar to `TESLA` [106].

They also presented a per-packet Byzantine detection protocol, called `OptimisticProtocol`, in a simply and elegant way. This protocol is an extension to `PERLMANd`, but addresses the inherent flaws. Basically, this protocol is also equivalent to `HSER`.

Finally, they developed a distributed detection protocol for aggregate traffic using either `PepperProbing` or `SaltProbing` approach to sample the traffic: Let  $\langle r_s, r_1, r_2, \dots, r_d \rangle$  be a path that is monitored. Every router along the path initiates a probing session with the destination router,  $r_d$ . At the end of the session,  $r_d$  sends all the traffic information,  $TI$ , that it has collected for all probing sessions to the source router  $r_s$ . Every intermediate router processes only the part of  $TI$  that it expects and then forwards  $TI$  to the upstream router. Then, source  $r_s$  collects fault reports from all the intermediate routers and detects a fault associated with a link  $\langle r_i, r_{i+1} \rangle$  if  $r_i$  and  $r_{i+1}$  report significantly different errors.

However this protocol still has high overhead in terms of state requirement at the intermediate routers. To provide secure communication to every router in the

network, the proposed protocol should be applied for every source and destination pair. Thus, a router needs to keep some information about each source and destination pair. This schema has  $O(R^2)$  complexity, where  $R$  is the number of routers in the network.

### 3.12 ZHANG: Secure routing in ad-hoc networks

The main difficulty for these detection protocols is to differentiate the packet losses due to congestion from malicious packet drops. Almost all detection protocols have tried to address this problem using a user-defined static threshold, and as we show in Section 6.4.3, it is impossible to find a threshold that can detect subtle attacks.

Congestion is a property of an interface. In order to determine congestion, the corresponding interface should be monitored. Zhang *et al.* [141] propose a distributed detection protocol, ZHANG, to detect malicious dropping of data packets by a bottleneck node in a wireless ad-hoc network.

They assume that the transmission patterns of each neighbor are stable and have a mean. A wireless node monitoring a neighbor estimates the senders' transmission rate as a Poisson process, which is used to compute a threshold value to predict the loss rate due to congestion. If the observed loss rate is significantly greater than the predicted threshold, then a failure is detected.

In terms of our specification, ZHANG is *strong-complete* and *accurate* with a *precision* of 2.

Our Protocol  $\chi$ , presented in Chapter 6, is also based on traffic validation per interface: dynamically inferring the number of congestive packet losses in a systematic manner.

## Chapter 4

# System Model and Specification

Our work proceeds from an informed, yet abstracted, model of how the network is constructed, the capabilities of the attacker, and the complexities of the traffic validation problem. In Section 4.1, we describe and motivate further assumptions underlying our model. In Section 4.2, we present the specification that we derived for traffic validation and distributed detection.

### 4.1 System Model

We consider a network to consist of individual routers interconnected via directional point-to-point links. In using this model, we purposely ignore several real-life complexities. For example, real routers are not homogeneous nodes, but in fact represent a collection of independent network interfaces which are themselves interconnected and are, in practice, addressed and controlled distinctly. We have eliminated this detail for the convenience of description; our analysis can easily accommodate this expanded model. Similarly, we have chosen to ignore the possibility of broadcast channels in our model since they are rare in today's wired networks and can be easily incorporated as collections of point-to-point links (although there may be opportunities for additional optimization in broadcast environments).

We assume that each router has sufficient data processing capability to generate traffic summaries describing the network traffic it is forwarding. For the most precise

summary functions this may involve touching all packet content. This is well within the capabilities of modern ASICs, but might require sampling in software implementations. We also assume that each router has sufficient computational capability to exchange and reconcile summaries with its neighbors. We discuss the issue of overhead further in Chapter 7, but in general, our algorithms are efficient and practical for existing router CPUs.

Within a network, we presume that packets are forwarded in a hop-by-hop fashion – each router following the directions of a local forwarding table. As well, we assume that these forwarding tables are updated via a distributed link-state routing protocol such as OSPF or IS-IS. This is critical, as we depend on the routing protocol to provide each node with a global view of the current network topology, which we assume to be *consistent*. Indeed, this assumption is valid in practice since studies of OSPF behavior have shown that in-network topology changes are extremely rare [115, 130]. When topology changes do occur there may be transient inconsistencies between routers that make it impossible to determine whether traffic is being forwarded correctly or not. Thus, if a compromised router frequently changes its connectivity, it may hide some forwarding attacks. For this reason, among others, any secure network infrastructure *also* requires software to detect security violations or anomalies in the control plane, as mentioned in Section 1.1.1.

Finally, we assume the administrative ability to assign and distribute shared keys to sets of nearby routers. This overall model is consistent with the typical construction of large enterprise IP networks or the internal structure of single ISP backbone networks, but is not well-suited for networks that are composed of multiple administrative domains using BGP.

We define a *path* to be a finite sequence  $\langle r_1, r_2, \dots, r_n \rangle$  of adjacent routers. Operationally, a path defines a sequence of routers that a packet can follow. We call the first router of the path the *source* and the last router its *sink*; together, these are called *terminal routers*. A path might consist of only one router, in which case the source and sink are the same.

An  $x$ -*path-segment* is a sequence of  $x$  consecutive routers that is a subsequence of a path. A *path-segment* is an  $x$ -*path-segment* for some value of  $x > 0$ . For example, if a network consists of the single path  $\langle a, b, c, d \rangle$  then  $\langle c, d \rangle$  and  $\langle b, c \rangle$  are both 2-*path-segments*, but  $\langle a, c \rangle$  is not because  $a$  and  $c$  are not adjacent.

We do not rely on source routing, as has been done by some work in the past, such as PERLMAN, HSER, ACL, PacketObituaries, and StealthProbing. We do assume some knowledge of the path that a packet will take, at least in the stable state. In link state protocols, this can be problematic, because they can take advantage of multiple paths with equal cost for load balancing purposes. Fortunately, the current generation of routers use a deterministic hash algorithm to spread the traffic load across available interfaces (eg, Cisco Express Forwarding [29] and Juniper routers with an Internet Processor ASIC [61]). Thus, a router can predict the path that a packet will take in the stable state based on its own routing tables and the hash functions.

We assume that attackers can compromise one or more routers in a network and may even compromise sets of adjacent routers as well. In general, we parameterize the strength of the adversary in terms of the maximum number of adjacent routers along a given path that can be compromised. However, we assume that between any two uncompromised routers that there is sufficient path diversity such that the malicious routers do not partition the network. Our protocols are also designed to detect anomalies between pairs of *correct* nodes and thus for simplicity we assume that a terminal router is not faulty with respect to traffic originating or being consumed by that router.

Finally, since link-state protocols operate by periodically measuring and disseminating information, we assume a synchronous system. At this level of abstraction, we can assume a synchronous network model of synchronized clocks and bounded message delays. Our goal is to extend the routing protocol to detect compromised routers. If the network behaves asynchronously for too long, then the routing tables will be updated, thereby changing the network topology. This assumption is common to all protocols we know of that have addressed the problem of detecting compromised routers.

The failure of a router is defined in terms of an interval of time, which in

practice corresponds with a period of time during which traffic measurements are made. Specifically, a router  $r$  is *traffic faulty* with respect to a path-segment  $\pi$  during  $\tau$  if  $\pi$  contains  $r$  and, during the period of time  $\tau$ ,  $r$  exhibits anomalous behavior with respect to forwarding data that traverses  $\pi$ . For example, router  $r$  can selectively alter, misroute, drop, reorder, or delay the data that flows through  $\pi$ , and it can fabricate new data to send along  $\pi$  such that the packets, if they were valid, would have been routed through  $\pi$ .

## 4.2 Specification

In this section, we present the specification that we derived for traffic validation and distributed detection.

### 4.2.1 Traffic Validation

The first problem we address is *traffic validation*: what information is kept about packet traffic and how it is used to determine that a router has been compromised.

We assume that a compromised router can arbitrarily alter its own forwarding behavior. For example, a compromised router can drop or modify selected (or all) packets, or divert them to other routers. However, given the distributed nature of packet forwarding, such bad behavior can be detected. For example, suppose traffic traverses a path  $\langle r_1, r_2, r_3 \rangle$  and that router  $r_2$  modifies this traffic. If routers  $r_1$  and  $r_3$  are not compromised, then one could simply compare what  $r_1$  sent to  $r_2$  and what  $r_3$  received from  $r_2$  to detect  $r_2$ 's behavior.

At an abstract level, we represent traffic validation mechanisms as a predicate  $TV(\pi, info(r_i, \pi, \tau), info(r_j, \pi, \tau))$  where:

- $\pi$  is a path-segment  $\langle r_1, r_2, \dots, r_x \rangle$  whose traffic is to be validated between routers  $r_i$  and  $r_j \in \pi$ .
- $info(r, \pi, \tau)$  is information about traffic that router  $r$  forwarded to along  $\pi$  over some time interval  $\tau$ .

- If routers  $r_i$  and  $r_j$  are not faulty, then  $TV(\pi, info(r_i, \pi, \tau), info(r_j, \pi, \tau))$  evaluates to *false* iff  $\pi$  contains a router  $r$  that was faulty in forwarding traffic along  $\pi$  during  $\tau$ , and router  $r$  is between routers  $r_i$  and  $r_j$  within  $\pi$ .

Implementing a traffic validation mechanism is a tricky engineering problem. The simplest, and most precise, representation of  $info(r, \pi, \tau)$  is a complete copy of the packets sent, with each packet tagged with the time it was forwarded. However, the storage requirements for buffering these packets, and the bandwidth consumed by resending them, make this approach impractical at best. In practice, implementing traffic validation is a tradeoff between accuracy and overhead. For example, sampling can be used to decrease overhead, but depending on how sampling is done and the duration of the attack, accuracy may be reduced. The overhead-accuracy tradeoff also depends on what limits one might place on the kind of bad behavior a compromised router can exhibit.

Similarly,  $TV$  could be implemented simply using equality;  $info(r_i, \pi, \tau) = info(r_j, \pi, \tau)$ . However, real networks occasionally lose packets due to congestion, reorder packets due to internal multiplexing, and corrupt packets due to interface errors. Consequently,  $TV$  needs to be somewhat more sophisticated to accommodate this abnormal, but non-malicious behavior. Thus, implementing traffic validation involves striking a balance between the acceptable number of false positives and false negatives.

Note that we have already made one engineering decision: we describe aggregate traffic rather than individual packets. This is in contrast to some prior work, e.g. PERLMAN, HERZBERG, HSER. While we compute state over each packet locally, limiting distributed reconciliation to traffic aggregates amortizes the communication and synchronization overhead (otherwise prohibitive) across many packets. This also makes it feasible to apply a threshold mechanism to distinguish between acceptable bad behavior (e.g. small amounts of packet loss and reordering) and malicious behavior.

### 4.2.2 Distributed Detection

The second problem is synchronizing the collection of traffic information and distributing the results for detection purposes. The solution to this problem is a protocol, and so we consider specifications of the problem as well as implementation.

Since routers collect the information upon which traffic validation is based, there will be some uncertainty in determining which router is faulty. For example, consider traffic that traverses a path containing the sequence of routers  $\langle r_1, r_2, r_3 \rangle$ . Suppose router  $r_3$  receives 10 packets from  $r_2$  and  $r_1$ 's traffic information states that  $r_1$  sent 20 packets to  $r_2$ . There's no way for  $r_3$  to determine whether  $r_2$  did indeed drop 10 packets, or whether  $r_1$  is misreporting the traffic.

We cast the problem as a failure detector with *accuracy* and *completeness* properties. This failure detector reports suspicions as path segments, which are sequences of adjacent routers. More specifically, a failure detector reports a path-segment  $\pi$  if it suspects a router in  $\pi$  is forwarding traffic along  $\pi$  in a faulty manner. A failure detector also has a *precision*, which is the maximum length of a path-segment it suspects.

Next, we present a formal derivation of this specification.

Due to the nature of the problem, our specification is more complex than the typical accuracy and completeness properties of an imperfect failure detector [23]). Since this detector is based on evaluating traffic collected over a period of time, we have the failure detector report pairs  $(r, \tau)$ , which means that  $r$  was suspected as being faulty during the time interval  $\tau$ . A perfect failure detector would implement the following two properties:

- *Accuracy (tentative)*: A failure detector is *accurate* if, whenever a correct router suspects  $(r, \tau)$ , then  $r$  was faulty during  $\tau$ .
- *Completeness (tentative)*: A failure detector is *complete* if, whenever a router  $r$  is faulty at some time  $t$ , then all correct routers eventually suspect  $(r, \tau)$  for some  $\tau$  containing  $t$ .

As noted above, though, our failure detector is based on traffic information collected by untrustworthy routers. This results in detections that are imprecise: we may not be able to pin down which router is compromised. So, we have the failure detector return a pair  $(\pi, \tau)$  where  $\pi$  is a path-segment and formalize it by defining a condition we call being *fault inclusive*. This also allows us to give more information: we restrict the detection to a router being faulty with respect to forwarding traffic along  $\pi$ .

- *a-Accuracy*: A failure detector is *a-Accurate* if, whenever a correct router suspects  $(\pi, \tau)$ , then  $|\pi| \leq a$  and some router  $r \in \pi$  was faulty in  $\pi$  during  $\tau$ .
- *a-FI Completeness (tentative)*: A failure detector is *a-FI Complete* if, whenever a router  $r$  is faulty at some time  $t$ , then all correct routers eventually suspect  $(\pi, \tau)$  for some path-segment  $\pi : |\pi| \leq a$  such that  $r$  was faulty in  $\pi$  at  $t$ , and for some interval  $\tau$  containing  $t$ .

Note that a faulty router can report an incorrect value for the traffic that traversed a path and can also alter this traffic. As defined in Section 2.2.1, the term *traffic faulty* indicates a router that alters traffic and the term *protocol faulty* indicates a router that misreports traffic. A *faulty* router is one that is traffic faulty, protocol faulty or both. As before, we will add the phrase “in  $\pi$ ” to indicate that the faulty behavior is with respect to traffic that transits the path  $\pi$ . Thus, the *a-Accuracy* requirement can result in a detection if a router is either protocol faulty or traffic faulty.

Distinguishing between protocol faulty and traffic faulty behavior is useful because, while it is important to detect routers that are traffic faulty, it isn't as critical to detect routers that are only protocol faulty: routers that are only protocol faulty are not altering the traffic flow. Hence, we weaken *a-FI Completeness*:

- *a-FI Completeness*: A failure detector is *a-FI Complete* if, whenever a router  $r$  is traffic faulty at some time  $t$ , then all correct routers eventually suspect  $(\pi, \tau)$  for some path-segment  $\pi : |\pi| \leq a$  such that  $r$  was traffic faulty in  $\pi$  at  $t$ , and for some interval  $\tau$  containing  $t$ .

One can't depend on faulty servers to detect faulty servers. Hence, failure detection is influenced more by the maximum number of adjacent faulty routers rather than the total number of faulty routers. So, we impose an upper bound  $AdjacentFault(k)$  on the number of adjacent faulty routers. For example, if  $AdjacentFault(3)$  holds, then there can be no more than 3 adjacent faulty routers in any path.

Making an  $AdjacentFault(k)$  assumption has an effect on failure detection. Allowing compromised routers to be adjacent complicates detection further, since the routers can cooperate to hide the evidence that a router is faulty. For example, assume that  $AdjacentFault(3)$  holds, and consider a path  $\langle r_1, r_2, r_3, r_4, r_5, r_6, r_7 \rangle$  in which  $r_3, r_4$  and  $r_5$  are faulty. Suppose that over an interval  $\tau$ ,  $r_1$  through  $r_5$  report having forwarded 100 packets that were to traverse this path, and  $r_6$  and  $r_7$  report receipt of only 20 such packets. Let  $r_1$  obtain these counters. It could be the case that  $r_4$  dropped the traffic, and  $r_3$  and  $r_5$  misreported the traffic in an effort to hide the fact that  $r_4$  is faulty. From  $r_1$ 's point of view, however, something is wrong with either  $r_5$  or  $r_6$ , since the counters indicate that traffic has disappeared between them. To satisfy  $a$ -FI Completeness,  $r_1$  needs to detect any router that could possibly have led to this traffic discrepancy. So, the failure detector at  $r_1$  could report that the path-segment  $\langle r_3, r_4, r_5, r_6 \rangle$  contains a faulty router, since if  $r_5$  is faulty then  $r_3$  and  $r_4$  could be as well, given  $AdjacentFault(3)$ . That is, we could have the failure detector report a  $(k+1)$ -length path-segment to accommodate the fact that  $AdjacentFault(k)$  holds.

Another way to accommodate this kind of scenario is to weaken  $a$ -FI Completeness. In the example just given, we could have  $r_1$  just suspect the path-segment  $\langle r_5, r_6 \rangle$ . A countermeasure protocol would stop routing data through this path, and if  $r_3$  or  $r_4$  continue to behave in a faulty manner, then they would be suspected later. We formalize this approach by defining a condition we call being *fault connected*. Given a path-segment  $\pi$  and an interval  $\tau$ , we say that a router  $r$  is fault connected to router  $s$  with respect to  $\pi$  if both  $r$  and  $s$  are in  $\pi$ , and all of the routers between  $s$  and  $r$  are faulty in  $\pi$  during  $\tau$ . Trivially, any router  $r$  is fault connected to itself even if  $r$  is correct. We then weaken  $a$ -FI Completeness again:

- *a-FC Completeness*: A failure detector is *a-FC Complete* if, whenever a router  $r$  is traffic faulty at some time  $t$ , then all correct routers eventually suspect  $(\pi, \tau)$  for some  $\pi : |\pi| \leq a$  and some  $\tau$  containing  $t$  such that there is a router  $r'$  that was faulty in  $\pi$  at time  $t'$  in  $\tau$  and is fault-connected to  $r$ .

The choice between the two completeness properties is not obvious. One would expect the precision obtainable under *a-FC Completeness* to be better, but it could increase the latency in detecting some traffic faulty routers. If the value of  $k$  for which *AdjacentFault(k)* holds is not large, then *a-FI Completeness* is probably a better choice because of its simplicity.

Since we are assuming arbitrarily faulty routers, we have to allow faulty routers to suspect correct routers. We address this issue in the countermeasure protocol: as with WATCHERS, the only suspicion that elicits a countermeasure is that when a router  $r$  suspects a path-segment that is adjacent to  $r$ . In this case, the countermeasure protocol will cause  $r$  not to route traffic along the suspected path-segment. A faulty router can already drop packets, and so allowing a faulty router to break links with its neighbor (as a result of faulty suspicions) adds no further disadvantage.

Note that any FI complete protocol is also FC complete. Finally, we use *completeness* as a synonym for FC completeness:

- *a-Completeness*: A failure detector is *a-Complete* if it is *a-FC Complete*.

We omit  $a$  in front of *a-Accuracy* and *a-Completeness*, whenever it is convenient, and we use *precision* instead:

- *Precision*: The maximum length of a path-segment that a failure detector suspects.

## Acknowledgement

Parts of Chapter 4 are reprints of the material as it appears in the IEEE Transactions on Dependable and Secure Computing, 2006, by Alper Tugay Mizrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage.

## Chapter 5

# Detecting Malicious Routers

In this chapter we present two concrete protocols that differ in accuracy, completeness, and overhead – one of which is likely inexpensive enough for practical implementation at scale. `Protocol  $\Pi_2$`  is based on traffic validation per path-segment nodes and is strong-complete and accurate with precision of 2. `Protocol  $\Pi_{k+2}$`  is based on traffic validation per path-segment ends and is strong-complete and accurate with precision of  $k + 2$ , where  $k$  is the maximum number of adjacent faulty routers. Next, we present a prototype system, called *Fatih*, that implements this approach on a PC router, and we describe our experiences with it. We show that *Fatih* is able to detect and isolate a range of malicious router actions with acceptable overhead and complexity. We believe our work is an important step in being able to tolerate attacks on key network infrastructure components.

### 5.1 `Protocol $\Pi_2$` : A Complete, Accurate Protocol with Precision 2

Given a complete and accurate traffic validation mechanism, an obvious approach is to have each router collect traffic information over some agreed-upon interval, and then use consensus algorithm to have all correct routers agree upon the traffic information. With this information, each correct router can determine which routers might be faulty.

For example, consider the 4-path-segment  $\pi = \langle r_1, r_2, r_3, r_4 \rangle$  where  $r_1$  and

$r_4$  are not faulty. Let  $info(r_i, \pi, \tau)$  be the traffic information that router  $r_i$  collects over during the agreed upon time interval  $\tau$  for the path-segment  $\pi$ . If at least one of the other routers is traffic faulty with respect to  $\pi$  during this interval, then  $TV(\pi, info(r_1, \pi, \tau), info(r_4, \pi, \tau))$  will be false. This implies that for some  $r_i$ ,  $TV(\pi, info(r_i, \pi, \tau), info(r_{i+1}, \pi, \tau))$  is false, which means that at least one of routers  $r_i$  and  $r_{i+1}$  is faulty. Since  $info(r_i, \pi, \tau)$  and  $info(r_{i+1}, \pi, \tau)$  were disseminated using consensus, all correct routers will know that at least one of  $\{r_i, r_{i+1}\}$  is faulty.

We use these observations to construct a 2-Accurate failure detector protocol. The first issue to address is over which paths a router should record information. Note that it will need to run an instance of the protocol for each path about which it records information. An obvious answer—over each data stream’s path—could result in an enormous set of paths. We can make the set smaller by having each router keep track of each  $x$ -path-segment of which it is a member, for some value of  $x$ . The number of  $x$ -path-segments can grow very quickly with increasing  $x$ , and so  $x$  should be as small as possible. It must be large enough so that any sequence of faulty routers will be surrounded by correct routers, since this is necessary in order to detect faulty behavior.

If we assume that  $AdjacentFault(k)$  holds, then the minimum value of  $x$  satisfying the above constraint is  $k + 2$ . Note that given source and destination may not be separated by at least  $k$  routers, and so a router also collects information about all paths of which it is a member whose length is less than  $k + 2$ .

The resulting `Protocol  $\Pi_2$`  is shown in Figure 5.1. In this protocol, each router  $r$  maintains the current topology  $T$  from which it derives its routing table. Each router  $r$  also maintains a set of path-segments  $P_r$  that contain  $r$  and that  $r$  monitors. A router  $r$  runs a thread for each path-segment in  $P_r$ .  $P_r$ , which is computed from  $T$ , includes all  $(k + 2)$ -path-segments containing  $r$  and all  $x$ -path-segments,  $3 \leq x < k + 2$  whose ends are terminal routers.

For each path-segment  $\pi \in P_r$ ,  $r$  synchronizes with the other routers in  $\pi$  and collects information for the same traffic passing through  $\pi$  for an agreed-upon interval  $\tau$ . Periodically,  $r$  sends that traffic information to all routers in  $\pi$  using consensus. This

```

failure detector()
cobegin
  for each path-segment  $\pi \in P_r$ :
     $suspect_r^\tau[\pi] = \{ \}$  // the set of suspicious path-segments in  $\pi$  that  $r$  detects during  $\tau$ 
    while (true) {
      synchronize with all routers in  $\pi$ ;
      collect traffic information  $info(r, \pi, \tau)$  about  $\pi$  for an agreed-upon interval  $\tau$ ;
      consensus ( $[info(1, \pi, \tau)]_1, [info(2, \pi, \tau)]_2, \dots, [info(|\pi|, \pi, \tau)]_{|\pi|}$ );
      // at this point all correct routers in  $\pi$  agree on the values of  $info(i, \pi, \tau)$ 
      for all  $i: 1 \leq i < |\pi|$ :
        if  $\neg TV(\pi, info(i, \pi, \tau), info(i + 1, \pi, \tau))$  then
           $suspect_r^\tau[\pi] = suspect_r^\tau[\pi] \cup \{ \langle i, i + 1 \rangle \}$ ;
          reliable broadcast ( $[info(i, \pi, \tau)]_i, [info(i + 1, \pi, \tau)]_{i+1}$ );
    }
coend

```

Figure 5.1: Protocol  $\Pi_2$ : A Complete, accurate protocol with precision 2.

data is digitally signed to prevent an attack during consensus. We use  $[x]_i$  to indicate that  $x$  is digitally signed by  $i$ .

Consider the traffic passing through a path-segment  $\pi$ . The traffic will be consistent — that is,  $TV(\pi, info(i, \pi, \tau), info(i + 1, \pi, \tau))$  will be *true* — for each pair of routers  $\langle i, i + 1 \rangle$  in  $\pi$  unless a discrepancy is introduced by a faulty router. In other words, if  $TV(\pi, info(i, \pi, \tau), info(i + 1, \pi, \tau))$  is *false* then at least one of the two routers  $i$  or  $i + 1$  is faulty. Note that it could either be traffic faulty or protocol faulty (because it reports traffic information that does not represent the actual traffic that transited during  $\tau$ ). In either case, a correct router  $r$  in  $\pi$  will put the 2-path-segment  $\langle i, i + 1 \rangle$  into the set  $suspect_r^\tau[\pi]$ , and reliably broadcast the evidence of the failure detection: ( $[info(i, \pi, \tau)]_i, [info(i + 1, \pi, \tau)]_{i+1}$ ). Upon receiving this information, all other correct routers will evaluate  $TV(\pi, info(i, \pi, \tau), info(i + 1, \pi, \tau))$  as *false* and detect the fault on the 2-path-segment  $\langle i, i + 1 \rangle$ .

Protocol  $\Pi_2$  is given in Figure 5.1. In Appendix B.2, we show that Protocol  $\Pi_2$  is **2-Accurate** and **2-FC Complete**.

### 5.1.1 Overhead

The cost of Protocol  $\Pi_2$  comes from the collection of traffic information and the overhead of synchronization and consensus.

**Collecting traffic information:** In the worst case, a router has to collect traffic information for each packet it has routed, independent of the size of  $P_r$ .

**Size of  $P_r$ :** The size of  $P_r$  indicates the number of different set of routers with which  $r$  synchronizes, maintains traffic information, and exchanges such information using consensus. By construction,  $|P_r|$  is  $O(k \times R^{k+1})$  where  $R$  is the maximum number of links incident on a router. In practice, though, we expect  $|P_r|$  to be much smaller. We have examined two network topologies, Sprintlink and EBONE, that were measured by the Rocketfuel project [120] and counted the number of distinct path-segments that a router monitors for different values of  $k$  in *AdjacentFault*( $k$ ) assumption.

The Sprintlink network consists of 315 routers and 972 links. On the average, a router has 6.17 links, and the maximum number of links that a router has is 45. In Figure 5.2, the maximum, average and median of  $|P_r|$  that a router is incident on and monitors is given for this network. The empirical results are much smaller than the theoretical upper bound of  $O(k \times 45^{k+1})$ . This is because, among other factors, a link state routing protocol chooses only one path between any two routers.

It is worthwhile to compare this overhead with WATCHERS, in which each router maintains 7 counters for each of its neighbors per each destination in the network. For this topology, implementing WATCHERS, a router maintains  $7 \times 6.17 \times 315 \approx 13,605$  counters on average; and the largest number of counters a router maintains is  $7 \times 45 \times 315 = 99,225$ .

Assuming the same weak threat model of WATCHERS, it is sufficient for a

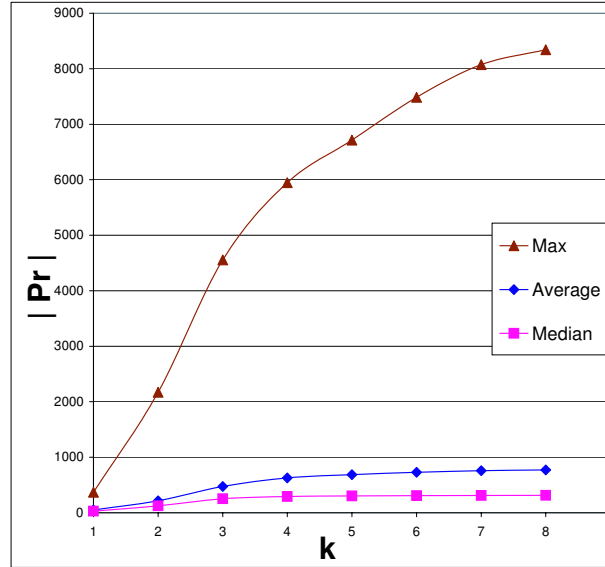


Figure 5.2: Based on  $AdjacentFault(k)$ ; maximum, average and median size of  $P_r$ , i.e. the number of path-segments monitored by an individual router in  $\text{Protocol } \Pi_2$ .

router, implementing  $\text{Protocol } \Pi_2$ , to maintain one counter for each path-segment in  $|P_r|$ . For this topology, assuming  $AdjacentFault(2)$ , a router maintains 216 counters on average; and the largest number of counters a router maintains is 2,172. If instead we have  $AdjacentFault(7)$ , these numbers become 758 and 8,073.

Examining the EBONE network, we obtain similar results. This is a smaller network: it consists of 87 routers and 161 links. On average, a router has 3.70 links, and the maximum number of links that a router has is 11.

**Synchronization, Consensus and Reliable Broadcast:** For each path-segment  $\pi$  in  $P_r$ , a router  $r$  synchronizes with all the routers in  $\pi$  to agree on when and for how long the next measurement interval  $\tau$  will be. Perfect synchronization would not be necessary in practice, since the traffic validation function  $TV$  could be written to accommodate a small skew. It would probably be more efficient, though, to have all the routers in the network synchronize with each other instead of having many more, smaller synchronization rounds.

Each router in path-segment  $\pi$  reaches consensus about the traffic information

over  $\pi$  during time interval  $\tau$ . To do so requires digital signatures of the traffic information, since otherwise the replication is not high enough for consensus to be solvable. Thus, there is an issue of *key distribution*, depending on the cryptographic tools that are used. Finally, there must be enough path connectivity among the routers to support consensus [72].

The final reliable broadcast will be done as part of the LSA distribution of link state protocol.

## 5.2 Protocol $\Pi_{k+2}$ : A Complete, Accurate Protocol with Precision

$k + 2$

Protocol  $\Pi_2$  has considerable requirements in terms of collecting traffic information, synchronization and consensus. These requirements can be avoided by making the detection less accurate.

**failure detector()**

**cobegin**

**for each** path-segment  $\pi \in P_r$ :

$suspect_r^\tau[] = \{ \}$  // the set of unreliable path-segments that r detects during  $\tau$

**while** (true) {

**synchronize** with the router  $r'$  at other end of  $\pi$ ;

**collect** traffic information  $info(r, \pi, \tau)$  about  $\pi$  for an agreed-upon interval  $\tau$ ;

**exchange**  $[info(r, \pi, \tau)]_r$  and  $[info(r', \pi, \tau)]_{r'}$  with  $r'$  through  $\pi$  within  $\mu$  timeout interval;

**if** (*exchange is failed* **or**  $\neg TV(\pi, info(r, \pi, \tau), info(r', \pi, \tau))$ ) **then** {

$suspect_r^\tau[] = suspect_r^\tau[] \cup \{ \langle \pi \rangle \}$ ;

**reliable broadcast** ( $[\pi]_r$ );

}

}

**coend**

Figure 5.3: Protocol  $\Pi_{k+2}$ : A Complete, accurate protocol with precision  $k + 2$ .

The idea is to apply  $TV$  just for the end nodes of each path-segment in  $P_r$ . For example, consider the 4-path-segment  $\pi = \langle r_1, r_2, r_3, r_4 \rangle$  where  $r_1$  and  $r_4$  are not faulty. Let  $info(r_1, \pi, \tau)$ ,  $info(r_4, \pi, \tau)$  be the traffic information that router  $r_1$  and  $r_4$  collect during  $\tau$ . If at least one of the other routers is traffic faulty with respect to  $\pi$  during this interval, then  $TV(\pi, info(r_1, \pi, \tau), info(r_4, \pi, \tau))$  will be false. In this case,  $r_1$  suspects  $\langle r_2, r_3, r_4 \rangle$ . Similarly  $r_4$  suspects  $\langle r_1, r_2, r_3 \rangle$ .

For this protocol, a router need not monitor as many path-segments as with Protocol  $\Pi_2$ . Instead, a router need only keep track of each  $x$ -path-segment, for which it is one of the end nodes, for some value of  $x$ . The number of  $x$ -path-segments can grow very quickly with increasing  $x$ , and so  $x$  should be as small as possible. It must be large enough so that any sequence of faulty routers will be surrounded by correct routers, as this is necessary in order to detect faulty behavior.

If we assume that  $AdjacentFault(k)$  holds, then the minimum value of  $x$  satisfying the above constraint is  $k + 2$ . However, monitoring only  $k + 2$ -path-segments is not sufficient. A trivial reason for this is that not all path-segments need be  $k + 2$  long. A more substantial reason is that compromised routers may hide another router's bad behavior. For example, given that  $AdjacentFault(2)$  holds, consider the 4-path-segment  $\pi = \langle r_1, r_2, r_3, r_4 \rangle$  where  $r_1$  and  $r_3$  are correct and  $r_2$  and  $r_4$  are faulty. In this case,  $r_1$  and  $r_4$  monitors  $\pi$  but  $r_4$  can hide the fact that  $r_2$  is traffic faulty by simply sending traffic information to  $r_1$  such that  $TV(\pi, info(r_1, \pi, \tau), info(r_4, \pi, \tau))$  holds. If  $r_1$  were to instead also monitor the path  $\langle r_1, r_2, r_3 \rangle$ , then  $r_1$  could detect  $r_2$ 's faulty behavior. So, it is necessary for a router  $r$  to monitor all  $x$ -path-segments for  $3 \leq x \leq k + 2$  of which  $r$  is an end.

For each path-segment  $\pi \in P_r$ ,  $r$  synchronizes with the other end router of  $\pi$  and collects information for the traffic passing through  $\pi$  during an agreed-upon interval  $\tau$ . Router  $r$  then exchanges digitally signed traffic information with the router  $r'$  on the other end. If the exchange operation fails within a pre-specified timeout interval  $\mu$ , or if  $r$  finds  $TV(\pi, info(r, \pi, \tau), info(r', \pi, \tau))$  is false, then there is at least one faulty router in  $\pi$  during  $\tau$ . In particular, either  $r'$  is protocol faulty or some router in  $\pi$  is traffic

faulty. So,  $r$  detects  $\pi - \langle r \rangle$ . However, when it announces this detection to the other routers, a correct router receiving this information suspects  $\pi$  since  $r$  might be faulty. For simplicity, we also have router  $r$  suspect  $\pi$ .

Protocol  $\Pi_{k+2}$  is given in Figure 5.3. In Appendix B.3, we show that Protocol  $\Pi_{k+2}$  is **(k+2)–Accurate** and **(k+2)–Complete**.

### 5.2.1 Overhead

Protocol  $\Pi_{k+2}$  is not very expensive. The main cost of the protocol is due to collecting traffic information.

**Collecting traffic information:** Assuming that a router uses the same values of  $\tau$ , traffic validation time interval, for all the path-segments in  $P_r$ , in the worst case a router has to collect traffic information for each packet it routes, which is independent of the size of  $P_r$ . The same holds for the previous Protocol  $\Pi_2$ . However, in Protocol  $\Pi_{k+2}$  this cost can be reduced by using sampling. For each  $\pi$  in  $P_r$ ,  $r$  can agree with the router  $r'$  on the other end on a random sampling pattern. The traffic they record on  $\pi$  would be determined by this pattern. Although the faulty routers in  $\pi$  could share their information on sampling and only attack the packets not being sampled by a faulty router, by construction there is a path-segment in  $P_r$  whose other end is not faulty, and so with the use of suitable encryption, any intermediate faulty routers will not know which packets are being sampled for traffic information. We don't know of a similar method of sampling that could be used for Protocol  $\Pi_2$ .

**Size of  $P_r$ :** The size of  $P_r$  indicates the number of routers with which  $r$  has to exchange traffic information.  $|P_r|$  is  $O(\min\{R^{k+1}, N\})$  where, as before,  $R$  is the maximum number of links incident to a router and  $N$  is the number of routers in the network. The second term,  $N$ , comes because a link state routing protocol chooses only one path between any two routers.

For the same Sprintlink(US) network topology that was analyzed in Sec-

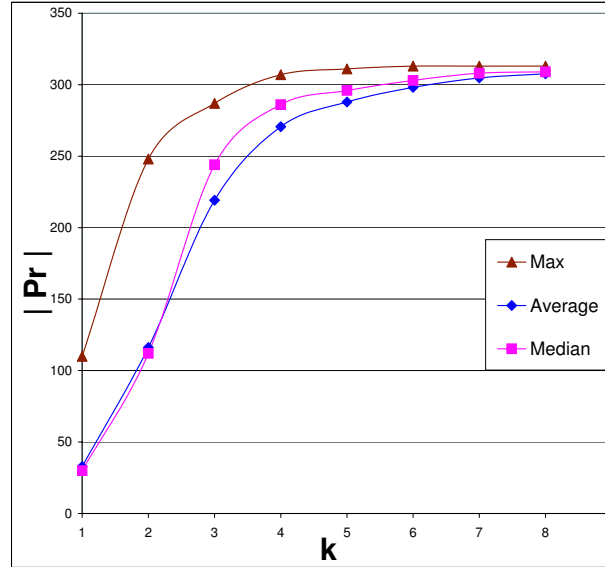


Figure 5.4: Based on  $AdjacentFault(k)$ ; maximum, average and median size of  $P_r$ , i.e. the number of path-segments monitored by an individual router in  $Protocol \Pi_{k+2}$ .

tion 5.1.1, the maximum, average and median of  $|P_r|$  that a router monitors in this protocol is given in Figure 5.4 for different values of  $k$  in  $AdjacentFault(k)$  assumption. As expected, these values are much lower than the theoretical upper bound, and are also much lower than the corresponding values for  $Protocol \Pi_2$ .

As a point of comparison, for the Sprintlink network, on average a router under WATCHERS would maintain approximately 13,600 counters, and the maximum number of counters that a router would maintain is 99,205. With our conservation of flow traffic summary function<sup>1</sup> for  $Protocol \Pi_{k+2}$ , a router maintains 232 counters on average and 496 in the worst case if  $AdjacentFault(2)$  holds. Even with the weak assumption of  $AdjacentFault(7)$ , each router maintains 616 counters on average and 626 in the worst case.

Finally, we argue that  $Protocol \Pi_2$  and especially  $Protocol \Pi_{k+2}$  are practically implementable, for smaller values of  $k$ .

<sup>1</sup>This requires two counters per path-segment, one for each direction.

**Synchronization, Consensus and Reliable Broadcast:** The synchronization requirements for `Protocol  $\Pi_{k+2}$`  are lower than for `Protocol  $\Pi_2$` . As for each path-segment  $\pi$  that a router  $r$  monitors,  $r$  needs to agree with only the other end router  $r'$  of  $\pi$ .

In order to exchange traffic information, neither Consensus nor the *good neighbor* condition of `WATCHERS` is required. The routers can use a pre-agreed upon round strategy to choose the values of  $\tau$ . Then the end routers can use the same path-segment they are monitoring to exchange traffic information. This is because if an intermediate router were to fail to forward the information, then one end would detect it, which would lead to the path-segment being suspected. Still, authentication is required to avoid impersonating attacks.

To prevent a faulty router impersonating a correct router, authentication of a failure detection announcement is required, which can be done with digital signatures. As with `Protocol  $\Pi_2$` , the final reliable broadcast can be done as part of the LSA distribution of link state protocol.

## 5.3 Fatih: Prototype System

We have implemented a prototype system, called Fatih, that incorporates our approach into a Linux 2.4-based router platform running OSPF. We have implemented a variety of traffic validation mechanisms mentioned in Section 2.4.1, including conservation of flow, content and order, as well as the `Protocol  $\Pi_{k+2}$`  distributed detection algorithm explained in Section 5.2.

### 5.3.1 System Architecture

The system architecture, shown in Figure 5.5, consists of five principal components:

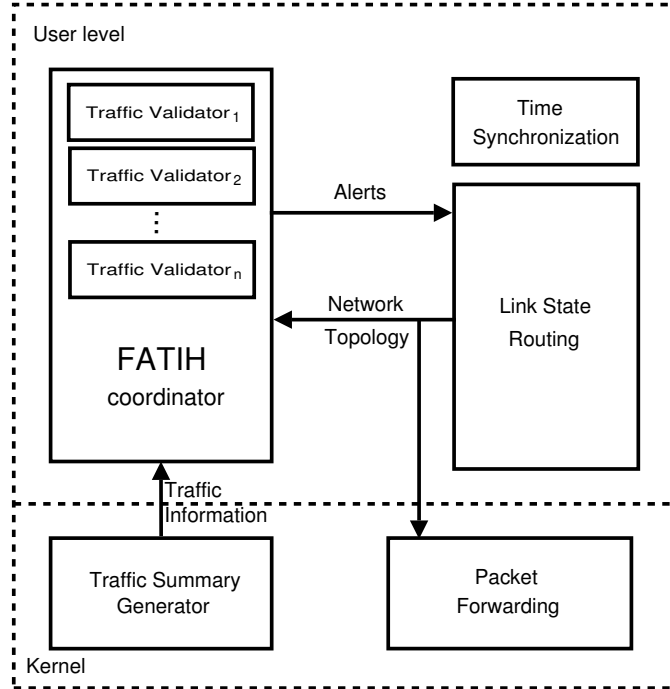


Figure 5.5: Fatih System Architecture

### FATIH Coordinator

This module implements the distributed detection algorithm, namely Protocol  $\Pi_{k+2}$ , and carries out the general scheduling and the communication with the Routing Daemon and the Traffic Summary Generator.

1. Based on the network topology exported by the Routing Daemon, the Coordinator decides which path-segments to monitor, depending on the system parameter  $k$  (Section 4.2.2). We have configured our prototype with  $k = 1$ , thus each router monitors all 3-path segments originating from itself. Beside simplicity, our implementation focuses on this point of the design space because it reflects the most common capabilities available to an attacker. For an adversary to compromise adjacent routers in a manner such that they attempt to conceal their actions (i.e.  $k > 1$ ) is considerably more difficult as this requires an adversary to modify the executable protocol code running on the routers.
2. The Coordinator receives information collected by the Traffic Summary Genera-

tor, determines the path being traversed and delivers the received information to the corresponding Traffic Validator modules.

3. Finally, the Coordinator schedules and synchronizes validation rounds among the Traffic Validators. In the current implementation, rounds are configured at 5 second intervals. A longer time interval requires more traffic summary state to be maintained, while a shorter time interval places more stringent synchronization requirements on the system.

*Traffic Validator:* For each monitored path-segment, there exists one corresponding Traffic Validator module, which keeps state for the traffic sent and received during the last time interval. At the end of each round it exchanges summary information with its corresponding peers and decides whether any discrepancies exceed acceptable limits. Messages between traffic validation modules on different routers is via an authenticated TCP connection (using manually configured shared keys in the current prototype). If a Traffic Validator does not receive any traffic information from its peer within a timeout interval, or if it decides that a traffic discrepancy is excessive, the path-segment is identified as “suspicious” to the Routing Daemon.

### **Traffic Summary Generator**

As described in Section 2.4.1, the Traffic Summary Generator updates traffic information with each forwarded packet. Depending on the underlying validation mechanism, the generator computes a packet fingerprint using the *UHASH* function [19] and associates a timestamp with it. The performance of this module is critical and therefore we have implemented it in the kernel to avoid unnecessary copies of packet contents. Traffic summaries are ultimately passed back to the Coordinator.

### **Link-State Routing Daemon**

As described in Section 4.1, Fatih cooperates with a standard link state routing protocol. The Routing Daemon, which is based on Zebra [38] in the current prototype,

implements the OSPF [90] protocol and manages link state announcements, shortest path computation and forwarding table calculation and installation. In addition, we have modified the protocol to incorporate input from Fatih. When the Routing Daemon receives an alert from the Coordinator (or via link-state updates from other routers), it recomputes new shortest path routes in order to avoid any suspicious path-segments identified. This alert is then flooded via the link-state update mechanism to allow other routers to exclude the suspicious path-segments as well.

However, it is not possible to reflect these routing changes in a single forwarding table update, because a router in the middle of a suspicious path-segment  $\pi$ , might need to forward traffic traversing a prefix of  $\pi$ , but destined for a path which is not a suffix of  $\pi$ . To allow this distinction, we exploit *policy based routing* to forward traffic using a combination of the source and destination addresses. The source address is used, effectively, to select the particular path-segment prefix upon which a packet has been delivered. The coordinator is kept abreast of routing changes so that it always knows which path-segments should be monitored, which peers to synchronize with, and so the source address can be efficiently mapped to a particular path-segment and forwarding table.

### **Packet Forwarding**

Linux supports policy-based routing through the use of multiple forwarding tables and an associated routing policy database. The database defines the criteria used to decide which forwarding table should be used to look-up a packet. For example, in our environment, a router maintains a distinct forwarding table for each detected suspicious path segment containing that router as a non end point.

### **Time Synchronization**

Fatih requires routers to have clocks synchronized closely enough so that there is not a significant disagreement over the intervals during which traffic information is collected. For our purposes, clocks that are synchronized within a few milliseconds are sufficient. We use NTP [80] to synchronize the routers clocks.

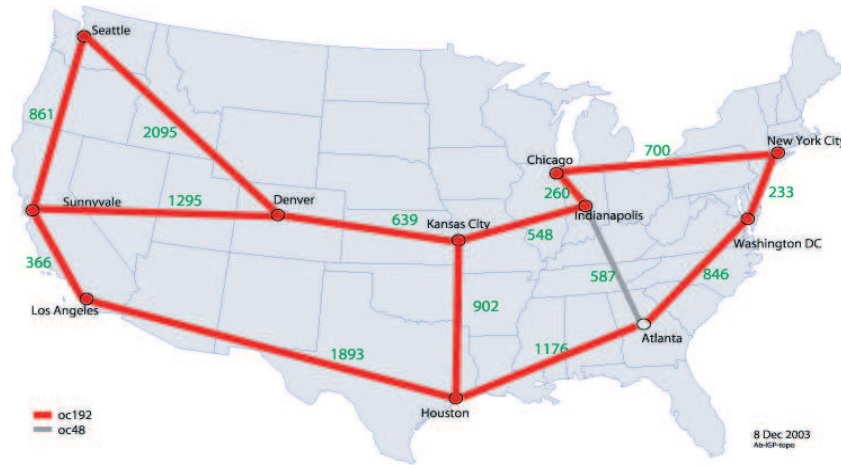


Figure 5.6: Abilene network topology.

### 5.3.2 Experiences

In this section, we describe the behavior of Fatih in a *simulated* network environment. While this methodology is sufficient to capture the gross behavior of Fatih in a distributed setting, it is not detailed enough to predict Fatih’s precise performance in an actual deployment. Similarly, our network traffic load is simulated as well and thus any end-to-end performance measurements could be misleading. Instead, we describe overhead on a “component” basis – fingerprint computation, router state, and synchronization overhead – and then explore how they might scale and be combined in the next section.

We have chosen a topology based on the Abilene network [1], Figure 5.6, because its structure, link delays, bandwidths and link-state metrics are all public. As well, the topology has sufficient connectivity to demonstrate the dynamics of Fatih during an attack.

We represent each Abilene Point of Presence (POP) as a single router and configure the link delay and routing metrics accordingly. Each of these routers is in turn emulated by a User-Mode Linux [127] virtual machine, configured with 64MB of memory and implementing the Fatih system architecture as described earlier. The routers are inter-connected through Ethernet bridging in the Linux host operating system, and

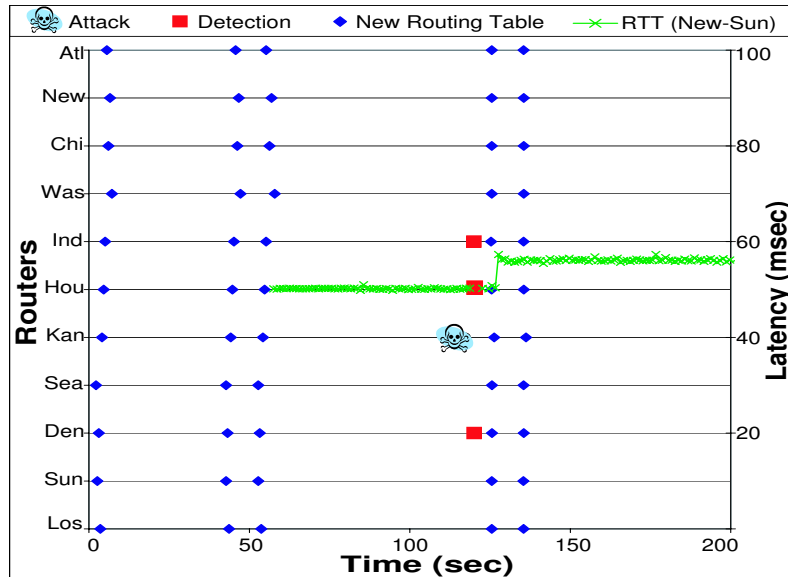


Figure 5.7: Fatih in progress.

modified to emulate configured link delays. The host system is a 2.6Ghz Pentium 4 server with 1GB of physical memory. Although our emulation testbed is incapable of processing the traffic volume of a real Internet backbone, it handles sufficient traffic to demonstrate Fatih's key behaviors.

Figure 5.7 demonstrates Fatih in progress. Time is shown on the  $x$ -axis in seconds, different routers are shown on the left  $y$ -axis, and the right  $y$ -axis is used for the latency measured between the *New York* and *Sunnyvale* during the experiment. At the beginning of the experiment, each router discovers its immediate neighbors, transmits and receives OSPF link state updates, and computes new routing tables with the most recent link state database. After roughly 55 seconds, all routers have agreed on a common network topology and a stable forwarding path is available between all pairs of routers.

At this point in time we inject a synthetic traffic load into the network and initiate round trip time(RTT) measurements between *New York* and *Sunnyvale*. Initially the forwarding path between these routers is  $\langle \text{Sunnyvale, Denver, Kansas City, Indianapolis, Chicago, New York} \rangle$  with a configured one-way latency of 25 ms in the configuration files. As expected the measured

round-trip time is roughly 50 ms.

At roughly 117 seconds, our simulated attacker compromises the *Kansas City* router and modifies its behavior such that 20% of its transit traffic is dropped or altered. We chose the *Kansas City* router as a victim because most inter-coastal traffic traverses it and is therefore an obvious target. Using the Fatih protocol, the path-segments  $\langle Denver, Kansas City, Indianapolis \rangle$ ,  $\langle Denver, Kansas City, Houston \rangle$  and  $\langle Houston, Kansas City, Indianapolis \rangle$  are all validated every  $\tau = 5$  seconds by their terminal routers. Thus, at the end of the current traffic validation round (about 3 seconds after the attack), *Denver*, *Houston* and *Indianapolis* detect that traffic through these monitored path segments is inconsistent and notify their OSPF routing daemons.

There are two parameters of the OSPF routing protocol that affect the following events. *OSPF\_delay\_time* is the time passed before computing a new routing table as a result of a triggering event (e.g. a new link-state update message or an alert, as in this case). *OSPF\_hold\_time* is the time passed before any consecutive routing table computations. These values default to 5 seconds and 10 seconds, respectively, in the Zebra OSPF implementation. As a result, an additional 15 seconds pass before the detected traffic inconsistency causes the associated path-segments to be removed from the routing topology. At roughly 135 seconds this process completes and the path between New York and Sunnyvale is changed to  $\langle Sunnyvale, Los Angeles, Houston, Atlanta, WashingtonDC, New York \rangle$ . This new path has a configured one-way latency of 28 ms, thus the measured RTT becomes 56 ms. Note that the *Kansas City* router continues to operate, but its neighboring routers will no longer forward traffic through it.

## Acknowledgement

Parts of Chapter 5 are reprints of the material as it appears in the IEEE Transactions on Dependable and Secure Computing, 2006, by Alper Tugay Mizrak, Yu-Chung

Cheng, Keith Marzullo and Stefan Savage.

## Chapter 6

# Detecting Congestive Losses

In this chapter, we consider the problem of detecting whether a compromised router is maliciously manipulating its stream of packets. In particular, we are concerned with a simple yet effective attack in which a router selectively drops packets destined for some victim. Unfortunately, it is quite challenging to attribute a missing packet to a malicious action because normal network congestion can produce the same effect. Modern networks routinely drop packets when the load temporarily exceeds a router's buffering capacity. Previous detection protocols have tried to address this problem with a user-defined threshold: too many dropped packets implies malicious intent. However this heuristic is fundamentally unsound; setting this threshold is, at best, an art and will certainly create unnecessary false positives or mask highly-focused attacks.

Several researchers have developed distributed protocols to detect such traffic manipulations, typically by validating that traffic transmitted by one router is received un-modified by another [21, 87]. However, all of these schemes – including our own – struggle in interpreting the *absence* of traffic. While a packet that has been modified in transit represents clear evidence of tampering, a missing packet is inherently ambiguous: it may have been explicitly blocked by a compromised router or it may have been dropped benignly due to network congestion. In fact, modern routers routinely drop packets due to bursts in traffic that exceed their buffering capacities, and the widely-used Transmission Control Protocol (TCP) is designed to *cause* such losses as part of

its normal congestion control behavior. Thus, existing traffic validation systems must inevitably produce false positives for benign events and/or produce false negatives by failing to report real malicious packet dropping.

In this chapter, we develop a compromised router detection protocol that dynamically infers the precise number of congestive packet losses that will occur. Once the congestion ambiguity is removed, subsequent packet losses can be safely attributed to malicious actions. We believe our protocol is the first to automatically predict congestion in a systematic manner and that it is necessary for making any such network fault detection practical.

In the remainder of this chapter, we evaluate options for inferring congestion, and then present the assumptions, specification and a formal description of a protocol that achieves these goals. We have evaluated our protocol in a small experimental network and demonstrate that it is capable of accurately resolving extremely small and fine-grained attacks.

## 6.1 Inferring Congestive Loss

In building a traffic validation protocol, it is necessary to explicitly resolve the ambiguity around packet losses. Should the absence of a given packet be seen as malicious or benign? In practice there are three approaches for addressing this issue:

- *Static Threshold.* Low rates of packet loss are assumed to be congestive, while rates above the threshold are deemed malicious.
- *Traffic modeling.* Packet loss rates are predicted as a function of traffic parameters and losses beyond the prediction are deemed malicious.
- *Traffic measurement.* Individual packet losses are predicted as a function of measured traffic load and router buffer capacity. Deviations from these predictions are deemed malicious.

### 6.1.1 Static Threshold

Most traffic validation protocols, including WATCHERS [21], Secure Traceroute [98] and our own work described in [86], analyze aggregate traffic over some period of time in order to amortize monitoring overhead over many packets. For example in [86], based on conservation of flow validation, `Protocol  $\Pi_{k+2}$`  maintains a set of packets counters at each router – two for each nearby router which performs validation. When a packet arrives at router  $r$  and is forwarded to a destination that will traverse a path-segment ending at router  $x$ ,  $r$  increments its outbound counter associated with router  $x$ . Conversely, when a packet arrives at router  $r$ , via a path-segment beginning with router  $x$ , it increments its inbound counter associated with router  $x$ . Periodically, router  $x$  sends a copy of its outbound counters to the associated routers for validation. Then a given router  $r$  can compare the number of packets which  $x$  claims to have sent to  $r$  with the number of packets it counts as being received from  $x$ , and it can detect the number of packet losses.

Thus, over some time window a router simply knows that out of  $m$  packets sent,  $n$  were successfully received. To address congestion ambiguity, all of these systems employ a pre-defined threshold: too many dropped packets implies some router is compromised. However, this heuristic is fundamentally flawed: how does one choose the threshold?

In order to avoid false positives, the threshold must be large enough to include the maximum number of possible congestive legitimate packet losses over a measurement interval. Thus, any compromised router that has the privilege to drop that many packets without being detected. Unfortunately, given the nature of the dominant Transmission Control Protocol (TCP), even small numbers of losses can have significant impacts. Subtle attackers can selectively target the traffic flows of a single victim and within these flows only drop those packets that cause the most harm. For example, losing a TCP SYN packet used in connection establishment has a disproportionate impact on a host because the retransmission timeout must necessarily be very long (typically 3 seconds or more). However, more generally seemingly minor attacks which cause TCP

timeouts can have similar effects – a class of attacks well described in [69].

All things considered, it is clear that the threshold mechanism is inadequate since it allows an attacker to mount vigorous attacks without being detected.

### 6.1.2 Traffic Modeling

Instead of using a static threshold, if the probability of congestive losses can be well modeled, then one could resolve ambiguities by comparing measured loss rates to the rates predicted by the model.

One approach for doing this is to analytically predict congestion as a function of individual traffic flow parameters, since TCP explicitly responds to congestion. Indeed, the behavior of TCP has been excessively studied [77, 97, 136, 22, 3]. A simplified<sup>1</sup> stochastic model of TCP congestion control yields the following famous square root formula:

$$B = \frac{1}{RTT} \sqrt{\frac{3}{2bp}}$$

where  $B$  is the throughput of the connection,  $RTT$  is the average round trip time,  $b$  is the number of packets that are acknowledged by one ACK, and  $p$  is the probability that a TCP packet is lost. The steady-state throughput of long-lived TCP flows can be described by this formula as a function of  $RTT$  and  $p$ .

This formula is based on a constant loss probability, which is the simplest model, but others have extended this work to encompass a variety of loss processes [3, 136, 59, 43]. Among these, the Bernoulli loss model is independent and identically distributed (iid), but given the measured burstiness of congestion losses, it is clear that these models are not satisfactory. The 2-state Markov model, also known as the Gilbert model, is able to capture the dependence between consecutive losses to some extent, but not for all situations. In fact, contradicting results have been reported by several researchers in trying to match these models to measured behavior: For example, [136]

---

<sup>1</sup>This formula omits many TCP dynamics such as timeouts, slow start, delayed acks, etc. More complex formulas taking these into account can be found in literature.

found that the Bernoulli model was accurate for 7 trace segments, while the 2-state Markov chain model was accurate for 10 segments out of 38 total trace segments. To model the rest of the traces a  $k^{\text{th}}$ -order Markov chain model was necessary. None of these models have been able to capture congestion behavior in all situations.

However, instead of attempting to infer congestion for individual flows, another approach is to statistically model congestion for the aggregate capacity of a link. In [5], Appenzeller *et.al.* explore the question of “How much buffering do routers need?”. A widely applied rule-of-thumb suggests that routers must be able to buffer a full delay bandwidth product. This controversial paper argues that due to congestion control effects, the rule-of-thumb is wrong, and the amount of required buffering is proportional to the square root of  $n$ , where  $n$  is the total number of TCP flows. To achieve this, the authors produced an analytic model of buffer occupancy as a function of TCP behavior.

In their analysis of desynchronized TCP flows, they model the bottleneck queue occupancy  $Q(t)$  at time  $t$  as a random process, and show that  $Q$  has a normal distribution with a standard deviation of

$$\sigma_Q = \frac{1}{3\sqrt{3}} \frac{2\overline{T}_p C + B}{\sqrt{n}} \quad (6.1)$$

where  $\overline{T}_p$  is the average propagation delay,  $C$  is the capacity of the bottleneck link,  $B$  is the maximum queue size of the bottleneck link, and  $n$  is the number of flows.

This result suggests that by counting the number of flows sharing the bottleneck link, congestion drops could be estimated with a probability  $p$ :

$$p = \frac{1 - \text{erf}\left(\frac{B/2}{\sqrt{2}\sigma_Q}\right)}{2} \quad (6.2)$$

We have evaluated this model thoroughly and verified that  $Q$  has a normal distribution in our experiments. However, the distribution does not meet the mean and standard deviation estimation suggested by Equation (6.1).

Though personal communication, we confirmed with the authors of [5] that

these formulas are a very rough approximation ignoring many details of TCP, such as timeouts, residual synchronization, retransmits, and a host of other effects. Thus, while this analysis is strong enough to model buffer *size*, it is not precise enough to accurately predict congestive losses.

### 6.1.3 Traffic Measurement

Our conclusion from previous experiments is that stochastic prediction of congestion is unlikely to provide a sufficiently precise prediction to be useful in a real system. Instead, we have turned to explicitly measuring the interaction of traffic load and buffer occupancy. To wit, for an output buffered FIFO router, congestion can be precisely predicted as a function of the inputs (the traffic rate delivered from all input ports destined to the target output port), the capacity of the output buffer, and the speed of the output link. Only if packet input rates from all sources exceed the output link speed long enough to fill the output buffer will a packet be lost. If such measurements are taken with high precision it should even be possible to predict individual packet losses. It is this approach that we consider further in the remainder of this chapter. We restrict our discussion to output buffered switches for simplicity although the same approach can be extended to input buffered switches or virtual output queues with additional adjustments (and overhead).

Because of some uncertainty in the system, we can not predict exactly which individual packets will be dropped. So, our approach is still based on thresholds. Instead of being a threshold on rate, it is a threshold on a statistical measure: the amount of confidence that the drop was due to a malicious attack rather than from some normal router function. To make this distinction clearer, we refer to the statistical threshold as the *target significance value*.

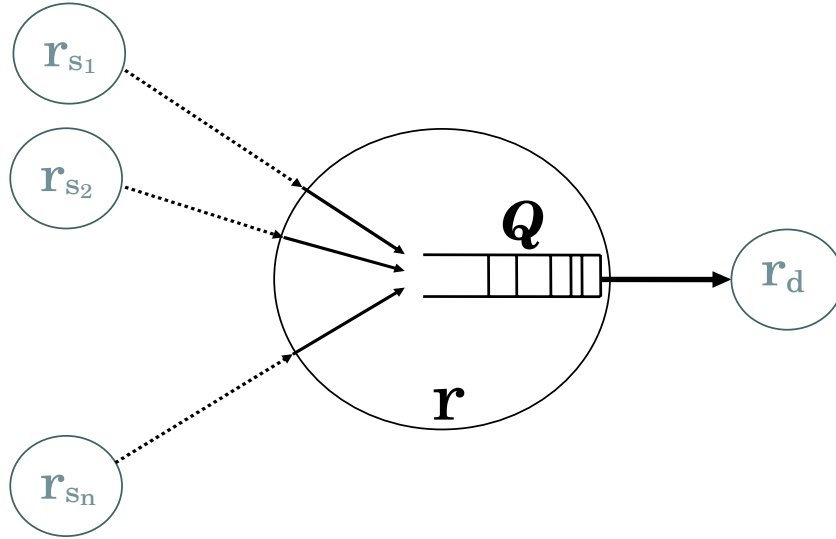


Figure 6.1: Validating the queue of an output interface.

## 6.2 Protocol $\chi$

We use the same system model presented in Section 4.1. Furthermore, we assume that the bandwidth, the delay of each link, and the queue limit for each interface are all known publicly. Attackers can compromise one or more routers in a network. However, for simplicity we assume in this chapter that adjacent routers cannot be *faulty*. Our work is easily extended to the case of  $k$  adjacent *faulty* routers.

Protocol  $\chi$  detects *traffic faulty* routers by validating the queue of each output interface for each router. Given the buffer size and the rate at which traffic enters and exits a queue, the behavior of the queue is deterministic. If the actual behavior deviates from the predicted behavior, then a failure has occurred.

We present the failure detection protocol in terms of the solutions of the distinct subproblems: traffic validation, information dissemination, and response.

### 6.2.1 Traffic Validation

The first problem we address is *traffic validation*: what information is collected about traffic and how it is used to determine that a router has been compromised.

Consider the queue  $Q$  in a router  $r$  associated with the output interface of link

$\langle r, r_d \rangle$ . See Figure 6.1. The neighbor routers  $r_{s_1}, r_{s_2}, \dots, r_{s_n}$  feed data into  $Q$ .

We denote with  $Tinfo(r, \mathbf{Q}_{dir}, \pi, \tau)$  the traffic information collected by router  $r$  that traversed path-segment  $\pi$  over time interval  $\tau$ .  $\mathbf{Q}_{dir}$  is either  $Q_{in}$ , meaning traffic into  $Q$ , or  $Q_{out}$ , meaning traffic out of  $Q$ . At an abstract level, we represent traffic a validation mechanism associated with  $Q$  as a predicate  $TV(Q, q_{pred}(t), S, D)$  where:

- $q_{pred}(t)$  is the predicted state of  $Q$  at time  $t$ .  $q_{pred}(t)$  is initialized to 0 when the link  $\langle r, r_d \rangle$  is discovered and installed into the routing fabric.  $q_{pred}$  is updated as part of traffic validation.
- $S = \{\forall i \in \{1, 2, \dots, n\} : Tinfo(r_{s_i}, Q_{in}, \langle r_{s_i}, r, r_d \rangle, \tau)\}$ , is a set of information about traffic coming into  $Q$  as collected by neighbor routers.
- $D = Tinfo(r_d, Q_{out}, \langle r, r_d \rangle, \tau)$  is the traffic information about the outgoing traffic from  $Q$  collected at router  $r_d$ .

If routers  $r_{s_1}, r_{s_2}, \dots, r_{s_n}$  and  $r_d$  are not protocol faulty, then  $TV(Q, q_{pred}(t), S, D)$  evaluates to *false* iff  $r$  was traffic faulty and dropped packets maliciously during  $\tau$ .

$Tinfo(r, \mathbf{Q}_{dir}, \pi, \tau)$  can be represented in different ways. We use a set that contains, for each packet traversing  $Q$ , a three-tuple that includes: a fingerprint of the packet, the packet's size and the time that the packet entered or exited  $Q$  (depending on whether  $\mathbf{Q}_{dir}$  is  $Q_{in}$  or  $Q_{out}$ ). For example, if at time  $t$  router  $r_s$  transmits a packet of size  $ps$  bytes with a fingerprint  $fp$ , and the packet is to traverse  $\pi$ , then  $r_s$  computes when the packet will enter  $Q$  based on the packet's transmission and propagation delay. Given a link delay  $d$  and link bandwidth  $bw$  associated with the link  $\langle r_s, r \rangle$ , the timestamp for the packet is  $t + d + ps/bw$ .

$TV$  can be implemented by simulating the behavior of  $Q$ . Let  $P$  be a priority queue, sorted by increasing timestamp. All the traffic information  $S$  and  $D$  are inserted into  $P$  along with the identity of the set ( $S$  or  $D$ ), from which the information came. Then,  $P$  is enumerated. For each packet in  $P$  with a fingerprint  $fp$ , size  $ps$ , and a

$$\begin{aligned}
c_{single} &= Prob(fp \text{ is maliciously dropped}) \\
&= Prob(\text{there is enough space in the queue to buffer } fp) \\
&= Prob(q_{act}(ts) + ps \leq q_{limit}) \\
&= Prob(X + q_{pred}(ts) + ps \leq q_{limit}) && \text{Random variable } X = q_{act}(ts) - q_{pred}(ts) \\
&&& \text{with mean } \mu \text{ and standard deviation } \sigma \\
&= Prob(X \leq q_{limit} - q_{pred}(ts) - ps) \\
&= Prob(Y \leq \frac{q_{limit} - q_{pred}(ts) - ps - \mu}{\sigma}) && \text{Random variable } Y = (X - \mu)/\sigma \\
&= Prob(Y \leq y_1) && y_1 = \frac{q_{limit} - q_{pred}(ts) - ps - \mu}{\sigma} \\
&= \frac{1 + erf(y_1/\sqrt{2})}{2} && erf \text{ is the error function.}
\end{aligned}$$

Figure 6.2: Confidence value for single packet loss test.

timestamp  $ts$ ,  $q_{pred}$  is updated as follows. Assume  $t$  is the time stamp of the packet evaluated prior to the current one:

- If  $fp$  came from  $D$ , then the packet is leaving  $Q$ :  $q_{pred}(ts) := q_{pred}(t) - ps$ .
- If  $fp$  came from  $S$  and ( $fp \in D$ ), then the packet  $fp$  is entering and will exit:  $q_{pred}(ts) := q_{pred}(t) + ps$ .
- If  $fp$  came from  $S$  and ( $fp \notin D$ ), then the packet  $fp$  is entering into  $Q$  and the packet  $fp$  would not be transmitted in the future:  $q_{pred}(ts)$  is unchanged, and the packet is *dropped*.
  - If  $q_{limit} < q_{pred}(t) + ps$ , where  $q_{limit}$  is the buffer limit of  $Q$ , then the packet is dropped due to congestion.
  - Otherwise, the packet is dropped due to malicious attack. Detect failure.

In practice, the behavior of a queue cannot be predicted with complete accuracy. For example, the tuples in  $S$  and  $D$  may be collected over slightly different intervals, and so a packet may appear to be dropped when in fact it is not (this is discussed in Section 6.3.1). Additionally, a packet sent to a router may not enter the queue

at the expected time because of short-term scheduling delays and internal processing delays.

Let  $q_{act}(t)$  be the actual queue length at time  $t$ . Based on the central limit theorem, our intuition tells us that the error,  $q_{error} = q_{act} - q_{pred}$ , can be approximated with a normal distribution. Indeed, this turns out to be the case as we show in Section 6.4. Hence, this suggests using a probabilistic approach. Doing so re-introduces a threshold in the form of a *confidence value*, but this can be more rationally chosen than the static thresholds described in Section 6.1.1.

We use two tests: one based on the loss of a single packet and one based on the loss of a set of packets.

### Single packet loss test

If a packet with fingerprint  $fp$  and size  $ps$  is dropped at time  $ts$  when the predicted queue length is  $q_{pred}(ts)$  then we raise an alarm with a confidence value  $c_{single}$ , which is the probability of the packet being dropped maliciously.  $c_{single}$  is computed as in Figure 6.2.

The mean  $\mu$  and standard deviation  $\sigma$  of  $X$  can be determined by monitoring during a learning period. We don't expect  $\mu$  and  $\sigma$  change much over time, because they are in turn determined by values that themselves don't change much over time. Hence, the learning period need not be done very often.

A malicious router is detected if the confidence value  $c_{single}$  is at least as large as a target threshold  $th_{single}$ .

### Combined packet losses test

The second test is useful when more than one packet is dropped during a round and the first test does not detect a malicious router. It is based on the well-known Z-test. Let  $L$  be the set of  $n > 1$  packets dropped during the last time interval. For the packets in  $L$ , let  $\overline{ps}$  be the mean of the packet sizes,  $\overline{q_{pred}}$  be the mean of  $q_{pred}(ts)$  (the predicted queue length) and  $\overline{q_{act}}$  be the mean of  $q_{act}(ts)$  (the actual queue length) over the times

the packets were dropped.

We test the hypothesis of “The packets are lost due to malicious attack”:  $\mu > q_{limit} - \overline{q_{pred}} - \overline{ps}$ . The Z-test score is:

$$z_1 = \frac{(q_{limit} - \overline{q_{pred}} - \overline{ps} - \mu)}{\sigma\sqrt{n}}$$

For the standard normal distribution  $Z$ , the probability of  $Prob(Z < z_1)$  gives the confidence value  $c_{combined}$  for the hypothesis. A malicious router is detected if  $c_{combined}$  is at least as large as a target threshold  $th_{combined}$ .

One can question using a Z-test in this way because the set of dropped packets are not a simple random sample. But, this test is used when there are packets being dropped and the first test determined that they were consistent with congestion loss. Hence, the router is under load during the short period the measurement was taken and most of the points, both for dropped packets and for non-dropped packets, should have a nearly-full  $Q$ . In Section 6.4 we show that the Z-test does in fact detect a router that is malicious in a calculated manner.

### 6.2.2 Distributed Detection

Since the behavior of the queue is deterministic, the traffic validation mechanisms detects *traffic faulty* routers whenever the actual behavior of the queue deviates from the predicted behavior. However, a faulty router can also be *protocol faulty*: it can behave arbitrarily with respect to the protocol by dropping or altering the control messages of `Protocol  $\chi$` . We mask the effect of protocol faulty routers using distributed detection.

Given  $TV$ , we need to distribute the necessary traffic information among the routers and implement a distributed detection protocol. Every outbound interface queue  $Q$  in the network is monitored by the neighboring routers and validated by a router  $r_d$  such that  $Q$  is associated with the link  $\langle r, r_d \rangle$ .

With respect to a given  $Q$ , the routers involved in detection are (as shown in Figure 6.1):

- $r_{s_*}$ , which send traffic into  $Q$  to be forwarded.
- $r$ , which hosts  $Q$ .
- $r_d$ , which is the router to which  $Q$ 's outgoing traffic is forwarded.

Each involved router has a different role, described below.

**Traffic Information Collection** Each router collects the following traffic information during a time interval  $\tau$ :

- $r_{s_*}$ : Collect  $Tinfo(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$ .
- $r$ : Collect  $Tinfo(r, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$ . This information is used to check the transit traffic information sent by the  $r_{s_*}$  routers.
- $r_d$ : Collect  $Tinfo(r_d, Q_{out}, \langle r, r_d \rangle, \tau)$ .

### Information Dissemination and Detection

- $r_{s_*}$ : At the end of each time interval  $\tau$ , router  $r_{s_*}$  sends  $[Tinfo(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)]_{r_{s_*}}$  that it has collected.  $[M]_x$  is a message  $M$  digitally signed by  $x$ . Digital signatures are required for integrity and authenticity against message tampering.<sup>2</sup>

1.  $r$ : Let  $\Delta$  be the upper bound on the time to forward traffic information.

(a) If  $r$  does not receive traffic information from  $r_{s_*}$  within  $\Delta$ , then  $r$  detects  $\langle r_{s_*}, r \rangle$ .

(b) Upon receiving  $[Tinfo(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)]_{r_{s_*}}$  router  $r$  verifies the signature and checks to see if this information is equal to its own copy  $Tinfo(r, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$ . If so, then  $r$  forwards it to  $r_d$ . If not, then  $r$  detects  $\langle r_{s_*}, r \rangle$ .

---

<sup>2</sup>Digital signatures can be replaced with message authentication codes if the secret keys are distributed among the routers.

At this point, if  $r$  has detected a failure  $\langle r_{s_*}, r \rangle$ , then it forwards its own copy of traffic information

$Tinfo(r, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$ . This is required by  $r_d$  to simulate  $Q$ 's behavior and keep the state  $q$  up to date.

2.  $r_d$ :

- (a) If  $r_d$  does not receive traffic information  $Tinfo(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$  originated by  $r_{s_*}$  within  $2\Delta$ , then it expects  $r$  to have detected  $r_{s_*}$  as faulty and to announce this detection through the response mechanism. If  $r$  does not do this, then  $r_d$  detects  $\langle r, r_d \rangle$ .
- (b) After receiving the traffic information forwarded from  $r$ ,  $r_d$  checks the integrity and authenticity of the message. If the digital signature verification fails, then  $r_d$  detects  $\langle r, r_d \rangle$ .
- (c) Collecting all traffic information, router  $r_d$  evaluates the  $TV$  predicate for queue  $Q$ . If  $TV$  evaluates to *false*, then  $r_d$  detects  $\langle r, r_d \rangle$ .

Note that dropping traffic information packets due to congestion can lead to false positives. Thus, the routers send this data with high priority. Doing so may cause other data to be dropped instead as congestion. Traffic validation needs to take this into account. It is not hard, but it is somewhat detailed, to do so in simulating  $Q$ 's behavior.

### 6.2.3 Response

Once a router  $r$  detects router  $r'$  as faulty,  $r$  announces the link  $\langle r, r' \rangle$  as being suspected. This suspicion is disseminated via the distributed link state flooding mechanism of the routing protocol. As a consequence, the suspected link is removed from the routing fabric.

Of course, a protocol faulty router  $r$  can announce a link  $\langle r, r' \rangle$  as being faulty, but it can do this for any routing protocol. And, in doing so, it only stops traffic from being routed through itself. Router  $r$  could even do this by simply crashing itself. To protect against such attack, the routing fabric needs to have sufficient path redundancy.

### 6.3 Analysis of Protocol $\chi$

In this section, we consider the properties and overhead of Protocol  $\chi$ .

#### 6.3.1 Accuracy and Completeness

In [87] we cast the problem of detecting compromised routers as a failure detector with *accuracy* and *completeness* properties. There are two steps in showing the accuracy and completeness of Protocol  $\chi$ :

- Showing that  $TV$  is correct.
- Showing that Protocol  $\chi$  is accurate and complete assuming that  $TV$  is correct.

As we assume that adjacent routers cannot be compromised in our threat model, we show in Appendix C that if  $TV$  is correct, then Protocol  $\chi$  is *2-accurate* and *2-complete*, where 2 indicates the length of detection: A link consisting of two routers is detected as a result. This assumption eliminates consorting faulty routers that collude together to produce fraudulent traffic information in order to hide their faulty behavior. This assumption can be relaxed to the case of  $k > 1$  adjacent faulty routers by monitoring every output interface of the neighbors  $k$  hops away and disseminating the traffic information to all neighbors within a diameter of  $k$  hops. This is the same approach that we used in [87], and it increases the overhead of detection.

We discuss traffic validation next.

#### 6.3.2 Traffic Validation Correctness

Any failure of detecting a malicious attack by  $TV$  results in a false negative, and any misdetection of legitimate behavior by  $TV$  results in a false positive.

Within the given system model of Section 4.1, the example  $TV$  predicate in Section 6.2.1 is correct. However, the system model is still simplistic. In a real router, packets may be legitimately dropped due to reasons other than congestion: for example, errors in hardware, software or memory, and transient link errors. Classifying these

as arising from a router being compromised might be a problem, especially if they are infrequent enough that they would be best ignored rather than warranting repairs on the router or link.

A larger concern is the simple way that a router is modeled in how it internally multiplexes packets. This model is used to compute timestamps. If the timestamps are incorrect, then  $TV$  could decide incorrectly. We hypothesize that a sufficiently accurate timing model of a router is attainable, but have yet to show this to be the case.

A third concern is with clock synchronization. This version of  $TV$  requires that all the routers feeding a queue have synchronized clocks. This requirement is needed in order to ensure that the packets are interleaved correctly by the model of the router.

The synchronization requirement is not necessarily daunting; the tight synchronization is only required by routers adjacent to the same router. With low level timestamping of packets, and repeated exchanges of time [8], it should be straightforward to synchronize the clocks sufficiently tightly.

Other representations of collected traffic information and  $TV$  that we have considered have their own problems of false positives and false negatives. It is an open question as to the best way to represent  $TV$ . We suspect any representation will admit some false positives or false negatives.

### 6.3.3 Overhead

#### Computing Fingerprints

The main overhead of Protocol  $\chi$  is in computing a fingerprint for each packet. This computation must be done at wire speed. Such a speed has been demonstrated to be attainable.

In our prototype, we implemented fingerprinting using UHASH [19]. [111] demonstrated UHASH performance of over 1Gbps on a 700Mhz Pentium III processor when computing a 4 byte hash value. This performance could be increased further with hardware support.

Network processors are designed to perform highly parallel actions on data packets [114]. For example, Feghali *et al.* [33] presented an implementation of well known private-key encryption algorithms on the Intel IXP28xx network processors to keep pace with a 10Gigabit/sec forwarding rate. Furthermore, Sanchez *et al.* [112] demonstrated hardware support to compute fingerprints at wire speed of high speed routers (OC-48 and faster).

### State Requirement

Let  $N$  be the number of routers in the network, and  $R$  be the maximum number of links incident on a router. `Protocol  $\chi$`  requires a router to monitor the path-segments that are at most two hops away. By construction, this is  $O(R^2)$ . State is kept for each of these segments. The  $TV$  predicate in Section 6.2.1 requires that a timestamp and the packet size be kept for each packet that traversed the path-segment. As a point of comparison, `WATCHERS` [21] requires  $O(RN)$  state, where each individual router keeps seven counters for each of its neighbors for each destination.

### Computing $TV$

The time complexity of computing  $TV$  depends on the size of the traffic information collected and received from the neighbors that are within 2 hops, and so it depends on the topology and the traffic volume on the network. If traffic information stores the packet fingerprints in order of increasing timestamps, then a straightforward implementation of traffic validation exists.

In our prototype, which is not optimized,  $TV$  computation had an overhead of between 15 to 20 milliseconds per validation round.

### Control Message Overhead

`Protocol  $\chi$`  collects traffic information and exchanges this information periodically using the monitored network infrastructure. Suppose each fingerprint and timestamp are both 4 bytes. Then, message overhead is 8 bytes per packet. If we assume

that the average packet size is 800 bytes, then the bandwidth overhead of `Protocol  $\chi$`  is 1%.

### **Clock Synchronization**

Similar to all previous detection protocols, `Protocol  $\chi$`  requires synchronization in order to agree on a time interval during which to collect traffic information. For a router  $r$ , all neighboring routers of  $r$  need to synchronize with each other to agree on when and for how long the next measurement interval  $\tau$  will be.

Clock synchronization overhead is fairly low. For example, external clock synchronization protocol NTP [80] can provide accuracy within 200 microseconds in local area networks. It requires two messages of size 90 bytes per transaction and the rate of transactions can be from once per minute to once per 17 minutes. [131] presented an internal clock synchronization protocol (RTNP) that maintains an accuracy within 30 microseconds by updating the clocks once every second.

### **Key Distribution**

To protect against protocol faulty routers tampering the messages containing traffic information, `Protocol  $\chi$`  requires digital signatures or message authentication codes. Thus, there is an issue of *key distribution*, and the overhead for this depends on the cryptographic tools that are used.

## **6.4 Experiences**

### **6.4.1 Simulation**

We have implemented `Protocol  $\chi$`  in ns2 [125], which is a widely deployed simulation environment for network protocols. This uses the *TV* predicate based on a timestamp and packet size for each packet. In our experiments, we have used the Abilene topology [1]. However, we configured the links as T3 links instead of OC192 links in order to decrease the burden on the simulator.

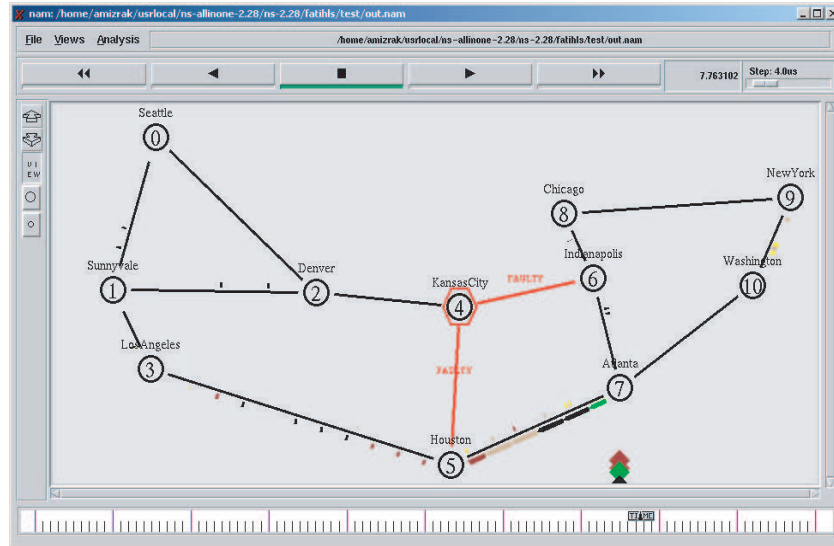


Figure 6.3: NS simulation Protocol  $\chi$ .

$\tau$ , the validation time interval, is set to 1sec.  $\Delta$ , the upper bound on the time to forward traffic information, is set to 150msec, which is reasonably large given that the maximum propagation delay of a link in the network is 11.09msec (between Los Angeles and Houston). We have chosen this aggressively large value in order to provide reliable delivery of the control messages via a mechanism consisting of acknowledgment, timeout and retransmission.

Figure 6.3 is a screen shot from an animation of one experiment. In the bootstrapping phase, routers discover their immediate neighbors and announce their connectivity to the network, and after receiving these link state updates, they compute new routing tables. This takes around 430msec.

At the end of each second, routers exchange traffic information corresponding the recent validation interval  $\tau = 1sec$ , and evaluate the  $TV$  predicate after  $2\Delta = 300msec$ .

We have created some background traffic in the network so that the KansasCity router gets congested. Later on, the KansasCity router is compromised and starts dropping one percent of the packets going to NewYork. In the following validation phase, this malicious behavior is detected by its neighbor Indianapolis, and the link

$\ell = \langle \text{KansasCity}, \text{Indianapolis} \rangle$  is announced to be suspicious by Indianapolis. Then  $\ell$  is removed from the routing fabric by the link state protocol.

Seattle and Sunnyvale routers now communicate with NewYork through  $\langle \dots, \text{LosAngeles}, \text{Houston}, \text{Atlanta}, \dots \rangle$ . However, the path from Denver to NewYork is still through KansasCity. Since KansasCity continues to attack those flows, in the following validation phase, Houston detects the malicious behavior of KansasCity and announces the link  $\langle \text{KansasCity}, \text{Houston} \rangle$  as suspicious, and as a result this link is also removed.

In the end,  $\langle \text{Denver}, \text{KansasCity} \rangle$  is still operational. However, Denver does not receive any more transit traffic to forward, and the communication between the uncompromised routers is safe.

We have implemented different kinds of attacks, including: i) selectively attacking specified flows from a given source to a given sink; ii) dropping connection setup packets (SYN, SYNACK) of the specified flows; iii) dropping a few packets of the specified flows so that TCP is forced to enter into the timeout state. Such packets occur when the TCP window is small, or after a fast retransmit or a timeout.

In our experiments, `Protocol  $\chi$`  has detected all of these attacks in the following validation phase while also distinguishing these attacks from congestion losses.

## 6.4.2 Network Emulation

We have implemented and experimented with `Protocol  $\chi$`  in the Emulab [133, 32] testbed. In our experiments, we used the simple topology shown in Figure 6.4. The routers were Dell PowerEdge 2850 PC nodes with a single 3.0 GHz 64-bit Xeon processor, 2GB of RAM, and they were running Redhat-Linux-9.0 OS software. Each router except for  $r_1$  was connected to three LANs to which user machines were connected. The links between routers were configured with 3 Mbps bandwidth, 20 msec delay, and 75000 byte capacity FIFO queues.

Each pair of routers share secret keys, furthermore integrity and authenticity against the message tampering is provided by message authentication codes.

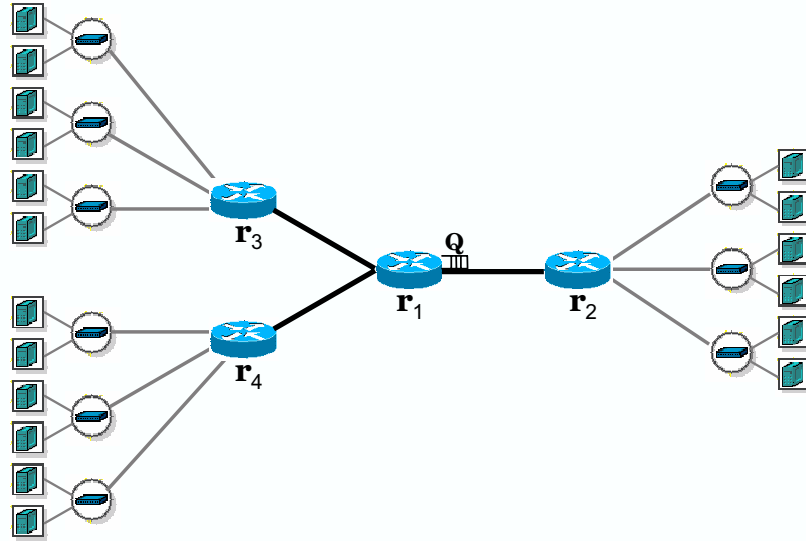


Figure 6.4: Simple topology.

The validation time interval  $\tau$  was set to 1 second and the upper bound on the time to forward traffic information  $\Delta$  was set to 300 milliseconds. At the end of each second, the routers exchanged traffic information corresponding the last validation interval, and evaluated the  $TV$  predicate after  $2\Delta = 600$  milliseconds. Each run in an experiment consisted of an execution of 80 seconds. During the first 30 seconds, we generated no traffic to allow the routing fabric to initialize. Then, we generated 45 seconds of traffic.

### Experiment 1: Protocol $\chi$ with no attack

We first investigated how accurately the protocol predicts the queue lengths of the monitored output interfaces. We considered the results for the output interface  $Q$  of  $r_1$  associated with the link  $\langle r_1, r_2 \rangle$ . Background traffic was created to make  $\langle r_1, r_2 \rangle$  a bottleneck. 20% of the bottleneck bandwidth was consumed by constant bit rate traffic, another 20% by short lived http traffic, and the rest by long lived ftp traffic.

The result of one run is shown in Figure 6.5(a).  $q_{pred}$  is the predicted queue length of  $Q$  computed by router  $r_2$  executing the Protocol  $\chi$ .  $q_{act}$ , which is the actual

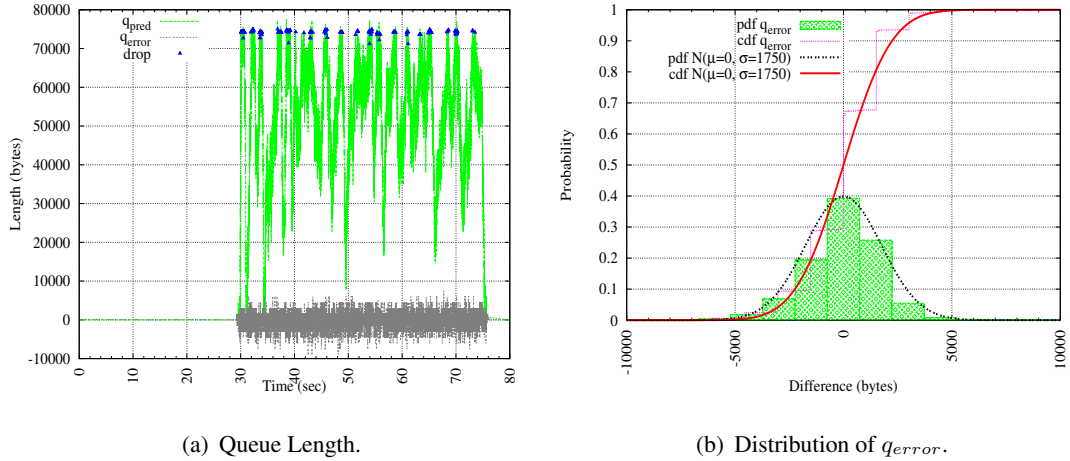


Figure 6.5: No attack.

queue length of  $Q$  recorded by router  $r_1$ , is not shown in the graph because it is so close to  $q_{pred}$ . Instead, the difference  $q_{error} = q_{act} - q_{pred}$ , is plotted; its value ranges approximately from -7500 bytes to 7500 bytes. Packet drops—all due to congestion—are marked with triangles.

Next, we examine the distribution of  $q_{error}$ . In Figure 6.5(b), the probability distribution and cumulative distribution functions of  $q_{error}$  are plotted. It is clustered around the multiples of 1500 bytes, since this is the maximum transmission unit and most frequent packet size of the traffic. Computing the mean,  $\mu$ , and the standard deviation,  $\sigma$ , of this data, the corresponding normal distribution functions are also shown in the graph. It turns out that the distribution of  $q_{error}$  can be approximated by a normal distribution  $N(\mu, \sigma)$ .

We expected many different causes to contribute to  $q_{error}$ : inaccurate clock synchronization, scheduling delays, internal processing delays, and so on. It turns out that scheduling and clock synchronization inaccuracy are the dominant factors. In terms of scheduling, all routers are running Linux with a programmable interval timer of 1024 Hz. This results in a scheduling quantum of roughly 1 millisecond. We verified the effect of the scheduling quantum by changing the frequency to 100Hz, and we observed that the variance of the distribution of  $q_{error}$  changed accordingly. For clock synchronization, we used NTP [80] to synchronize the routers' clocks, but it takes a long time for

the NTP daemon to synchronize the routers' clocks to within a few milliseconds. So, we used a different strategy: once a second we reset each router's clock to the NTP server's clock. This resulted in the clocks being synchronized to within 0.5 msec. Finally, the processing delay of the packets within a router is typically less than 50 microseconds. So, it does not introduce significant uncertainty as compared to other factors.

### Experiment 2: False positives

In the second experiment, we first ran a training run to measure the mean and standard deviation of  $q_{error}$ . We found  $\mu = 0$  and  $\sigma = 1750$ . We then ran `Protocol  $\chi$`  under a high traffic load for more than one hour, which generated more than half a million packets. Approximately 4,000 validation intervals occurred within this run, and approximately 16,000 packets were dropped due to congestion. Choosing  $th_{single} = 0.999$  and  $th_{combined} = 0.9$ , there were eight false positives generated by the single packet drop test and two false positives generated by the combined packet drop test. Both results are lower than one would expect, given the number of samples. We suspect that the lower false positive rate for the single packet drop test is because the distribution of  $q_{error}$  is not truly a normal distribution, and the lower false positive rate for the combined packet drop test is because the test is not done on a simple random sample. We are investigating this further. In all of the subsequent experiments, we used the same mean, standard deviation, and two thresholds given here.

### Experiment 3: Detecting attacks

We then experimented with the ability of `Protocol  $\chi$`  to detect attacks. In these experiments, the router  $r_1$  is compromised to attack the traffic selectively in various ways, targeting two chosen *ftp* flows. The duration of the attack is indicated with a line bounded by diamonds in the figures, and a detection is indicated by a filled circle.

For the first attack, the router  $r_1$  was instructed to drop 20% of the selected flows for 10 seconds. Predicted queue length and the confidence values for each packet drop can be seen in Figure 6.6(a) and Figure 6.6(b). As shown in the graph, during the

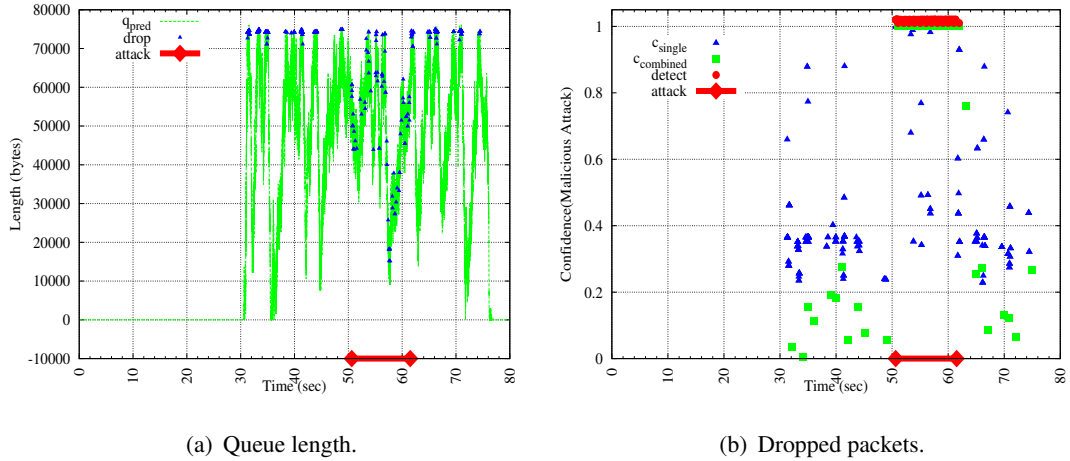


Figure 6.6: Attack 1: Drop 20% of the selected flows.

attack, Protocol  $\chi$  detected the failure successfully.

In the second attack, router  $r_1$  was instructed to drop packets in the selected flows when the queue was at least 90% full. Protocol  $\chi$  was able to detect the attack and raised alarms, as shown in Figure 6.7.

Next, we increase the threshold for which  $r_1$  attacks to 95%. No single drop test has enough confidence to raise an alarm because all of the drops are very close to the  $q_{limit}$ . However, Protocol  $\chi$  raised alarms for the combined drops test. Even though few additional packets were dropped, the impact on the TCP flows of this attack was significant. Both attacked flows' bandwidth usage dropped more than 35%, and their share was used by the other flows.

Last of all, we looked in the SYN attack which would prevent a selected host establishing a connection with any server: The router  $r_1$  was instructed to drop all SYN packets from a targeted host, which tries to connect to an ftp server. In Figure 6.9, five SYN packets, which are marked with circles, are maliciously dropped by  $r_1$ . Except for the second SYN packet drop, all malicious drops raised an alarm. The second SYN is dropped when the queue is almost full, and so the confidence value is not significant enough to differentiate it from the other packet drops due to congestion.

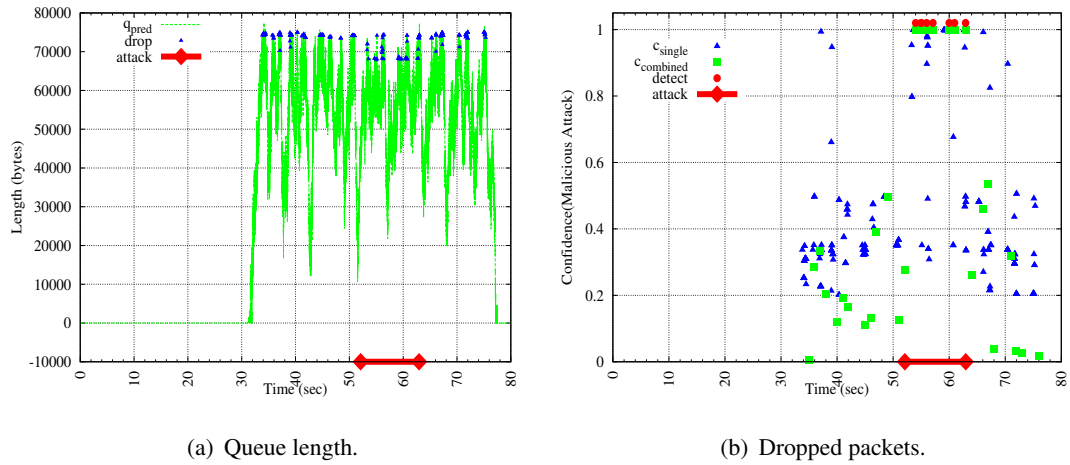


Figure 6.7: Attack 2: Drop the selected flows when the queue is 90% full.

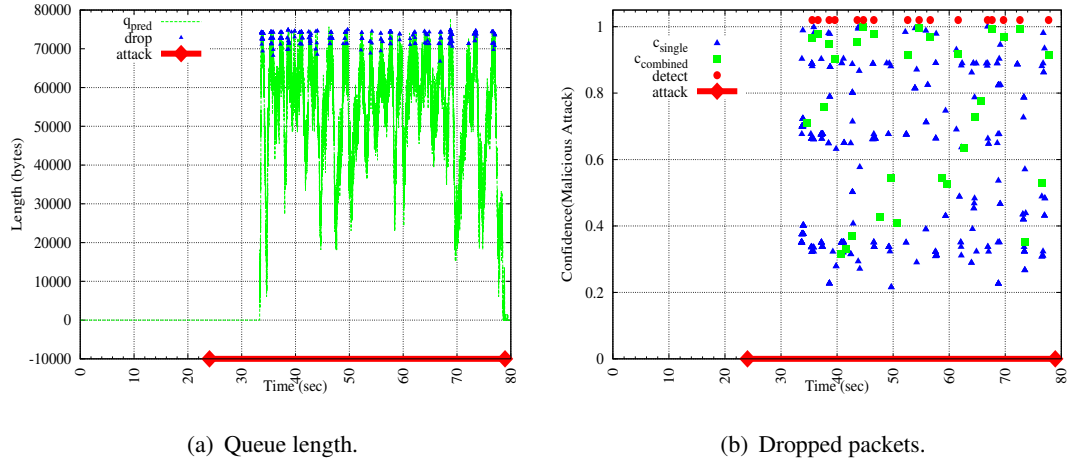


Figure 6.8: Attack 3: Drop the selected flows when the queue is 95% full.

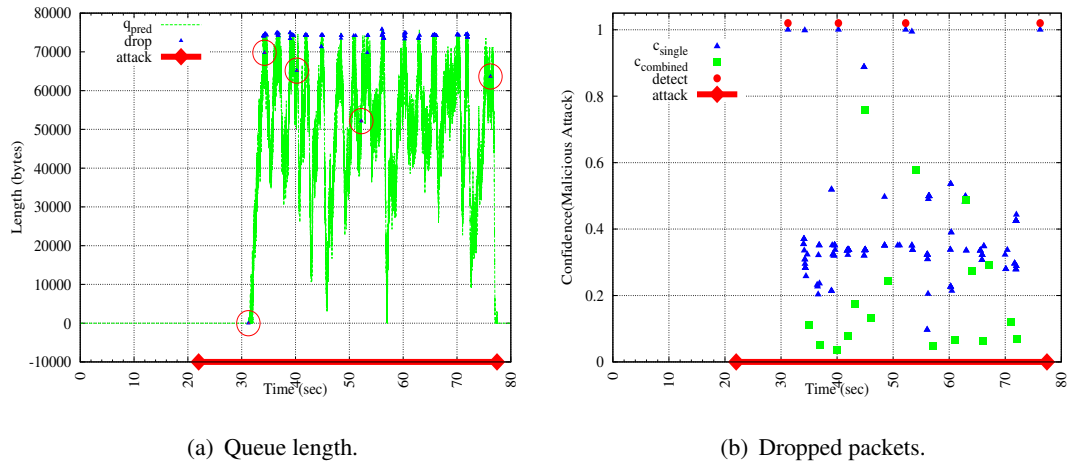


Figure 6.9: Attack 4: Target a host trying to open a connection by dropping SYN packets.

### 6.4.3 Protocol $\chi$ vs. Static threshold

We argued earlier the difficulties of using static thresholds of dropped packets for detecting malicious intent. We illustrate this difficulty with the run shown in Figure 6.7. Recall that during this run, the router dropped packets only when the output queue was at least 90% full. Before time 52, the router behaved correctly, and 2.1% of the packets were dropped due to congestion. During the time period from 52 to 64, the router maliciously dropped packets, but only 1.7% of the packets were dropped (some due to congestion and some due to the attack). This may seem counterintuitive: fewer packets were dropped due to congestion during the period that the queues contained more packets. Such a nonintuitive behavior doesn't happen in every run, but the dynamics of the network transport protocol led to this behavior in the case of this run. So, for this run, there is no static threshold that can be used to detect the period during which the router was malicious. A similar situation occurs in the highly-focused SNY attack of Figure 6.9.

In contrast, Protocol  $\chi$  can detect such malicious behaviors because it measures the router's queues, which are determined by the dynamics of the network transport protocol. Protocol  $\chi$  can have false positive and false negative detections, but the probability of such false detections can be controlled by setting a significance level for the statistical tests upon which Protocol  $\chi$  is built. A static threshold can not be used in the same way.

## 6.5 Non-deterministic Queuing

As described, our traffic validation technique assumes a deterministic queuing discipline on each router: *first in first out* (FIFO) with tail-drop. While this is a common model, in practice, real router implementations can be considerably more complex – involving switch arbitration, multiple layers of buffering, multicast scheduling, etc. Of these, the most significant for our purposes is the non-determinism introduced by active queue management (AQM), such as *random early detection* (RED) [35], *proportional*

*integrator* (PI) [50], and *random exponential marking* (REM) [9]. In this section, we describe how `Protocol  $\chi$`  can be extended to validate traffic in AQM environments. We focus particularly on RED, since this is the most widely-known and widely-used of such mechanisms.<sup>3</sup>

RED was first proposed by Floyd and Jacobson in the early 1990s to provide better feedback for end-to-end congestion control mechanisms. Using RED, when a router’s queue becomes full enough that congestion may be imminent, a packet is selected at random to signal this condition back to the sending host. This signal can take the form of a bit marked in the packet’s header and then echoed back to the sender – *Explicit Congestion Notification* (ECN) [34, 109] – or can be indicated by dropping the packet.<sup>4</sup> If ECN is used to signal congestion, then `Protocol  $\chi$` , as presented in Section 6.2, works perfectly. If not, then RED will introduce non-deterministic packet losses that may be misinterpreted as malicious activity.

In the remainder of this section, we explain how RED’s packet selection algorithm works, how it may be accommodated into our traffic validation framework, and how well we can detect even small attacks in a RED environment.

### 6.5.1 Random Early Detection

RED monitors the average queue size,  $q_{avg}$ , based on an exponential weighted moving average:

$$q_{avg} := (1 - w)q_{avg} + w \cdot q_{act} \quad (6.3)$$

where  $q_{act}$  is the actual queue size, and  $w$  is the weight for a low-pass-filter.

RED uses three more parameters:  $q_{min}^{th}$ , minimum threshold;  $q_{max}^{th}$ , maximum threshold; and  $p_{max}$ , maximum probability. Using  $q_{avg}$ , RED dynamically computes a dropping probability in two steps for each packet it receives. First, it computes a interim

<sup>3</sup>Although RED is universally *implemented* in modern routers, it is still unclear how widely it is actually used.

<sup>4</sup>ECN-based marking is well-known to be a superior signaling mechanism [74, 68]. However, while ECN is supported by many routers (Cisco and Juniper) and end-systems (Windows Vista, Linux, Solaris, NetBSD, etc) it is generally not enabled by default and thus it is not widely deployed in today’s Internet [103, 79].

probability,  $p_t$ :

$$p_t = \begin{cases} 0 & \text{if } q_{avg} < q_{min}^{th} \\ p_{max} \frac{q_{avg} - q_{min}^{th}}{q_{max}^{th} - q_{min}^{th}} & \text{if } q_{min}^{th} < q_{avg} < q_{max}^{th} \\ 1 & \text{if } q_{max}^{th} < q_{avg} \end{cases}$$

Further, the RED algorithm tracks the number of packets,  $cnt$ , since the last dropped packet. The final dropping probability,  $p$ , is specified to increase slowly as  $cnt$  increases:

$$p = \frac{p_t}{1 - cnt \cdot p_t} \quad (6.4)$$

Finally, instead of generating a new random number for every packet when  $q_{min}^{th} < q_{avg} < q_{max}^{th}$ , a suggested optimization is to only generate random numbers when a packet is dropped [35]. Thus, after each RED-induced packet drop, a new random sample,  $rn$ , is taken from a uniform random variable  $R = Random[0, 1]$ . The first packet whose  $p$  value is larger than  $rn$  is then dropped, and a new random sample is taken.

## 6.5.2 Traffic Validation for RED

Much as in Section 6.2.1, our approach is to predict queue sizes based on summaries of their inputs from neighboring routers. Additionally, we track how the predicted queue size impacts the likelihood of a RED-induced drop and use this to drive two additional tests: one for the uniformity of the randomness in dropping packets and one for the distribution of packet drops among the flows.<sup>5</sup> In effect, the first test is an evaluation of whether the *distribution* of packet losses can be explained by RED and tail-drop congestion alone, while the second evaluates if the particular *pattern* of losses (their assignment to individual flows) is consistent with expectation for traffic load.

<sup>5</sup>Consistent with our assumption that the network is under a single administrative domain (Section 4.1, we assume that all RED parameters are known).

Packet	$fp_1$	$fp_2$	$fp_3$	$fp_4$	$fp_5$	$fp_6$	$fp_7$	$fp_8$	$fp_9$	$\dots$	$\dots$	$\dots$	$fp_n$
Drop probability	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_6$	$p_7$	$p_8$	$p_9$	$\dots$	$\dots$	$\dots$	$p_n$
Outcome	$TX$	$TX$	$DR$	$TX$	$TX$	$TX$	$TX$	$DR$	$TX$	$\dots$	$\dots$	$\dots$	$TX$
Random number	$m_1$	$m_2$		$m_3$									

Figure 6.10: A set  $n$  packets. Each packet  $fp_i$  is associated with a drop probability  $p_i$  and the outcome is either transmitted(TX) or dropped(DR) based on the random number generated during the last packet drop.

### Testing the uniformity packet drops

In Figure 6.1, router  $r_d$  monitors the queue size of router  $r$  and detects whether each packet is dropped or transmitted. Given the RED algorithm and the parameters,  $r_d$  now can estimate  $q_{avg}$ , the average queue size in Formula 6.3;  $cnt$ , the count since the last dropped packet; and finally  $p$ , the dropping probability in Formula 6.4 for each packet as in Figure 6.10. All of these computations are deterministic and based on observed inputs.

The router  $r$  drops a packet  $fp_i$  if its  $p_i$  value exceeds the random number  $m_x$  that it generated at the most recent packet drop. So,  $r_d$  expects that  $m_x$  is between  $p_{i-1}$  and  $p_i$ . For example in Figure 6.10:

- $fp_3$  is dropped:  $p_2 < m_1 < p_3$ .
- $fp_8$  is dropped:  $p_7 < m_2 < p_8$ .

Since each packet drop should be a sample of a uniform random distribution, we can detect deviations from this process via statistical hypothesis testing. In particular, we use the Chi-Square test to evaluate the hypothesis that the observed packet losses are a good match for a uniform distribution [73]. Once the *Chi-Square* value<sup>6</sup> is computed, then the corresponding critical value can be used as the confidence value  $C_{randomness}$  to reject the hypothesis, which means the outcome is a result of non-uniform distribution

<sup>6</sup> *Chi-Square* =  $\sum_{i=1}^k \frac{(O_i - E_i)^2}{E_i}$ , where  $O_i$  is the observed frequency of bin  $i$ ;  $E_i$  is the expected frequency of bin  $i$ ; and  $k$  is the number of bins.

and/or a detection of malicious activity. Thus, a malicious router is detected if the confidence value  $c_{randomness}$  is at least a target significance level  $s_{randomness}^{level}$ .

### Testing the distribution of packet drops among flows

One of the premises of RED [35] is that the probability of dropping a packet from a particular connection is proportional to that connection's bandwidth usage. We exploit this observation to evaluate whether the particular pattern of packet losses – even if not suspicious in their overall number – is anomalous with respect to per-flow traffic load.

This test requires per-flow state in order to count the number of received packets and dropped packets per-flow during  $q_{min}^{th} < q_{avg} < q_{max}^{th}$ . Once again, we use the Chi-Square test to evaluate the distribution of packet losses to flows.<sup>7</sup> Once the *Chi-Square* value is computed, the corresponding critical value can be used as the confidence value  $c_{drop/flow}$  to reject the hypothesis, which means that the distribution of packet drops among the flows is not as expected. A malicious router is detected if the confidence value  $c_{drop/flow}$  is at least a target significance level  $s_{drop/flow}^{level}$ .

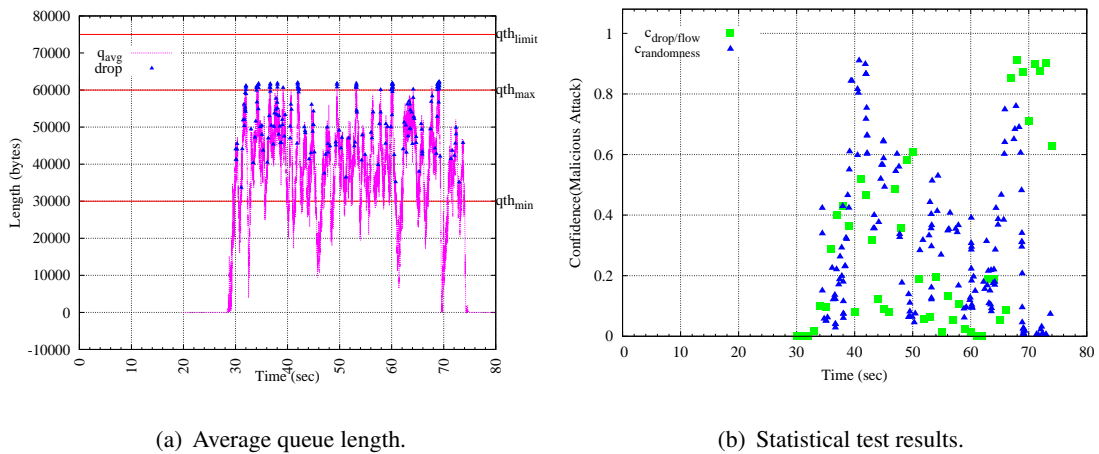


Figure 6.11: Without attack.

<sup>7</sup>Short lived flows with a few tens of packets are ignored unless the drop rate is 100%. Otherwise, a few packet drops from a short lived flow lead to false detection.

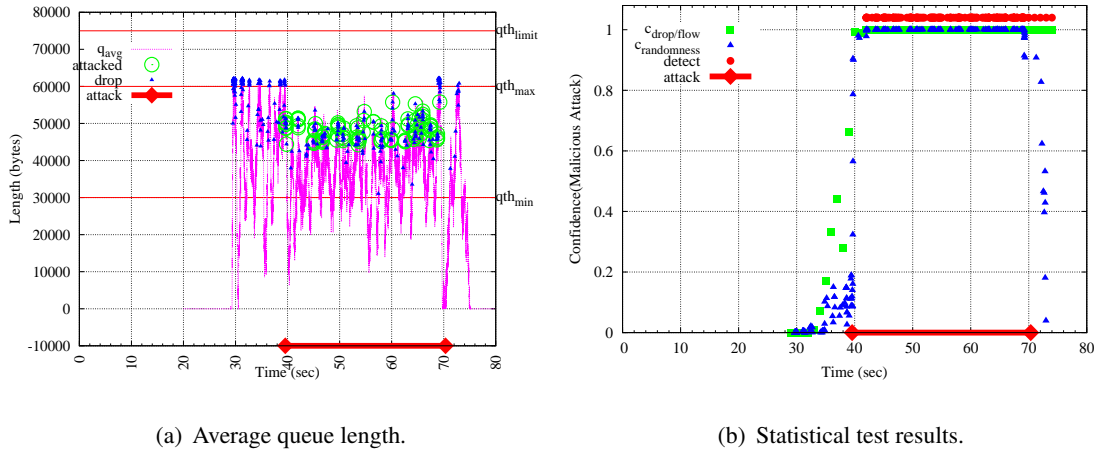


Figure 6.12: Attack 1: *Drop the selected flows when the average queue size is above 45,000 bytes.*

### 6.5.3 Experiences

We have experimented with `Protocol  $\chi$`  with this new traffic validation in a RED environment using the same setup as presented in Section 6.4. The capacity of the queue,  $q^{th}_{limit}$ , is 75000 bytes. In addition, the RED parameters, as in Section 6.5.1, are configured as following: the weight for the low-pass-filter is  $w = 0.5$ , the minimum threshold is  $q^{th}_{min} = 30,000$  bytes, the maximum threshold is  $q^{th}_{max} = 60,000$  bytes, and the maximum probability is  $p_{max} = 0.02$ .<sup>8</sup>

For the *packet drop uniformity test*, a window of 30 packet drops is used. The *distribution of packet drops to flows test* examines a window of 15 seconds. Experimentally we find that smaller windows lead to false positives, but larger windows do not improve the results notably. A more sophisticated version of our algorithm could adapt the window size in response to load in order to ensure a given level of confidence.

#### Experiment 1: False positives

The result of one run is shown in Figure 6.11(a).  $q_{avg}$  is the predicted average queue length of  $Q$  computed by router  $r_2$ . Packet losses are also marked with triangles.

<sup>8</sup> Setting the parameters is inexact engineering. We used the guidelines presented in [35] and/or our intuition in selecting these values.

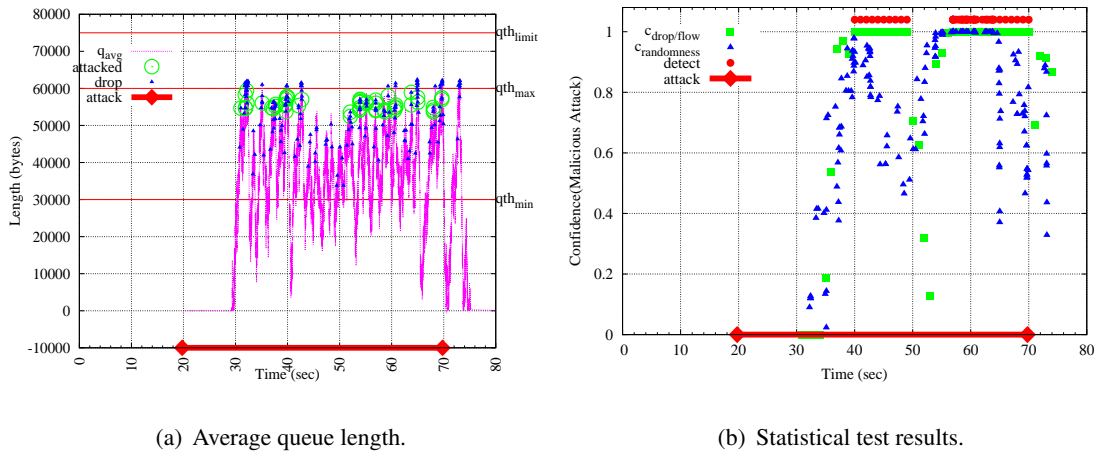


Figure 6.13: Attack 2: Drop the selected flows when the average queue size is above 54,000 bytes.

The corresponding confidence values can be seen in Figure 6.11(b).

We executed Protocol  $\chi$  under high traffic load for more than half an hour. With significance levels aggressively chosen at  $s_{randomness}^{level} = 0.999$  and  $s_{drop/flow}^{level} = 0.999$ , we did not observe any false positives.

## Experiment 2: Detecting attacks

Next we examined how effectively Protocol  $\chi$  detects various attacks. In these experiments, router  $r_1$  is compromised to attack the traffic selectively in various ways, targeting *ftp* flows from a chosen subnet. The duration of the attack is indicated with line bounded by diamonds in the figures, and a detection is indicated by a filled circle.

For the first attack, router  $r_1$  drops the packets of the selected flows for 30 seconds when the average queue size computed by RED is above 45,000 bytes. The predicted average queue size and the confidence values can be seen in Figure 6.12. As shown in the graph, during the attack, Protocol  $\chi$  detects the failure successfully.

As queue occupancy grows, the RED algorithm drops packets with higher probability and thus provides more “cover” for attackers to drop packets without be-

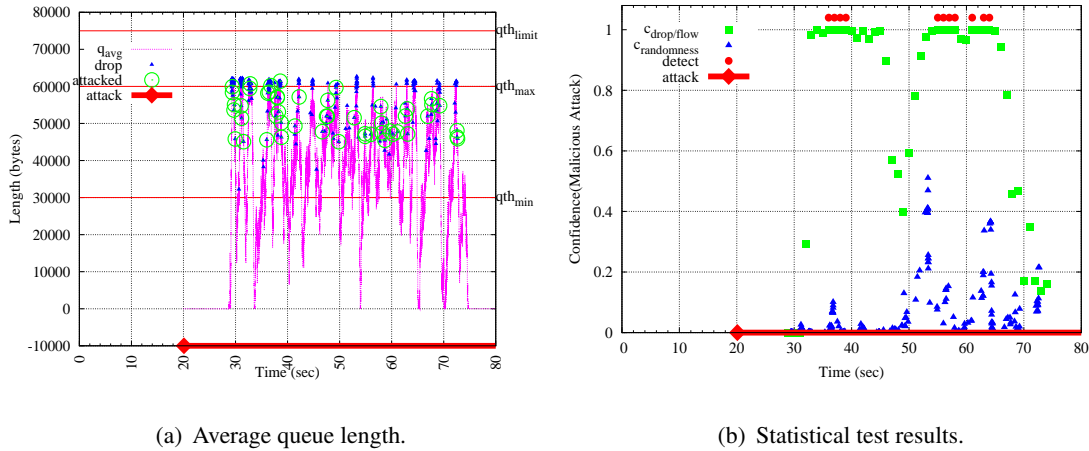


Figure 6.14: Attack 3: Drop 10% of the selected flows when the average queue size is above 45,000 bytes.

ing detected. We explore this property in the second attack, in which router  $r_1$  was instructed to drop packets in the selected flows when the average queue was at least 54,000 bytes, which is very close to the maximum threshold,  $q^{th}_{max} = 60,000$  bytes. As shown in Figure 6.13, Protocol  $\chi$  was still able to detect the attack and raised alarms, except between 50 and 56 seconds. The reason is that between 44 and 50 seconds, the compromised router did not drop any packets maliciously.

In the third and fourth attacks, we explore a scenario in which a router  $r_1$  only drops a small percentage of the packets in the selected flows. For example, during the third attack 10% of packets are dropped (see Figure 6.14) and 5% during the fourth attack (see Figure 6.15). Even though relatively few packets are dropped, the impact on TCP performance is quite high, reducing bandwidth by between 30% and 40%. Since only a few packets are maliciously dropped, the *packet drop uniformity test* does not detect any anomaly. However, since these losses are focused on a small number of flows, they are quickly detected using the second test.

Finally, we explained a highly-selective attack in which the router  $r_1$  was instructed to only drop TCP SYN packets from a targeted host, which tries to connect to an ftp server. In Figure 6.16, four SYN packets, which are marked with circles, are maliciously dropped by  $r_1$ . Since *all* the observed packets of the attacked flow are dropped,

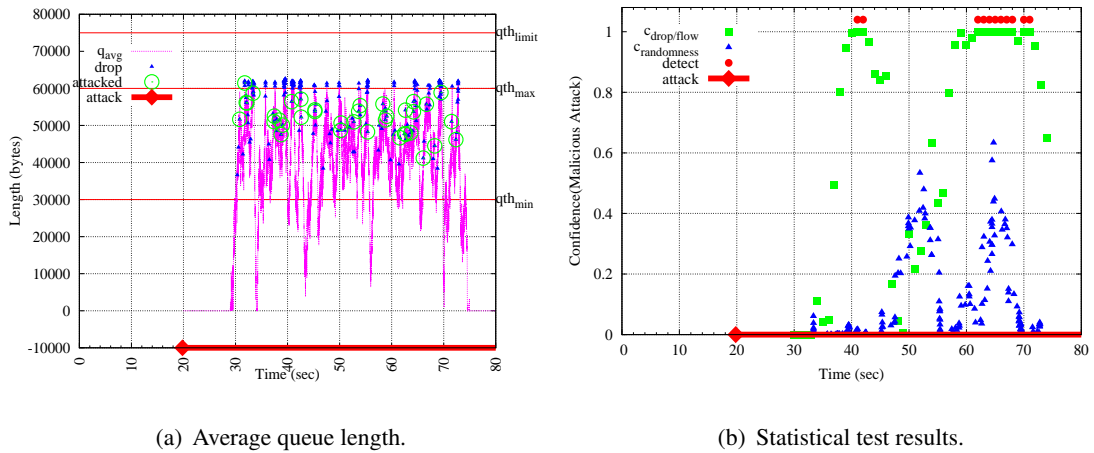


Figure 6.15: Attack 4: Drop 5% of the selected flows when the average queue size is above 45,000 bytes.

which is statistically unexpected given the RED algorithm, Protocol  $\chi$  still raises an alarm.

## Acknowledgement

Parts of Chapter 6 are reprint of the material as it appears in UCSD Technical Report, CS2007-0889, 2007, by Alper Tugay Mızrak, Keith Marzullo and Stefan Savage.

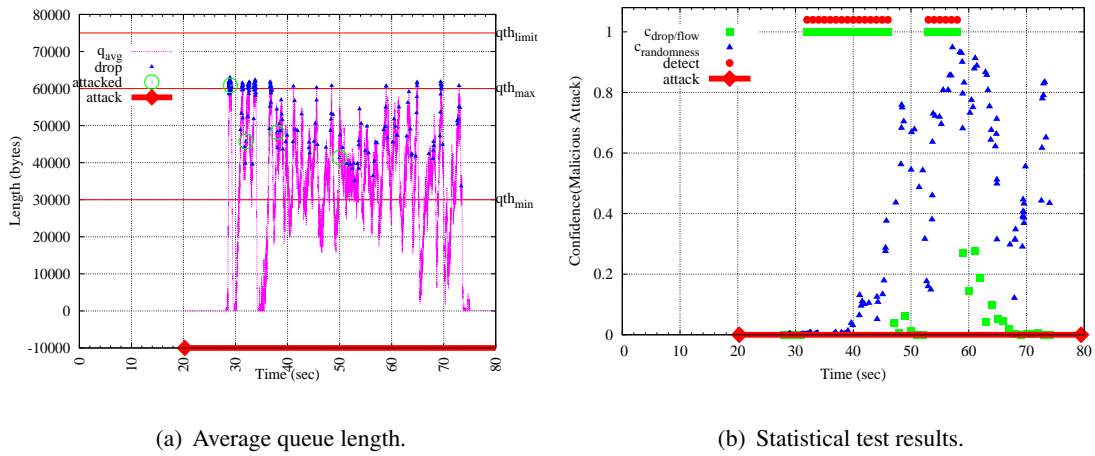


Figure 6.16: Attack 5: Target a host trying to open a connection by dropping SYN packets.

## Chapter 7

# Analysis of the Protocols

In this chapter, we consider the overhead of the proposed protocols and various issues that need to be addressed.

### 7.1 Computing Fingerprints

Any protocol that validates conservation of content for the purpose of detecting packet alteration requires the computation of a fingerprint for each packet. To be practical, computing a fingerprint must have a low overhead. One possibility for fingerprinting is CRC32. This is already computed by most network interfaces at line rate and is a good uniform hash function. However, it is reversible and an adversary could create modified packets that have the same hash. In some circumstances, such as a compute-limited adversary, it might serve as a good fingerprinting function.

We implemented fingerprinting using UHASH which is an unkeyed version of the UMAC algorithm [19]. This algorithm allows a trade-off between security and performance and is designed to support parallel implementations. [19, 111] demonstrated UHASH performance of over 1Gbps on a 700Mhz Pentium III processor computing a 4 byte hash value and delivering a  $2^{-30}$  forging probability (which is more than sufficient for our application). In hardware, of course, this performance could be increased even further.

Between software and hardware are emerging network processors which are

designed to perform highly parallel actions on data packets [114]. Recently, Feghali *et al.* [33] described an implementation of DES [95], AES [96],<sup>1</sup> and SHA-1 on the Intel IXP28xx network processors, that was able to keep pace with a 10Gigabit/sec forwarding rate. Since DES and AES encryption algorithms are significantly more expensive to compute than functions such as UHASH, we can infer that it is possible to compute these packet hashes at speeds of up to 10Gbps.

However, if there are insufficient computational resources, one can easily tradeoff accuracy for overhead by subsampling which packets are considered. As in Duffield and Grossglauser’s Trajectory Sampling [31], if the same random hash function is used to subsample packets at each end of a path-segment, then each router should observe the same subset of packets. Further, each pair of routers is free to select such hash functions independently and need not rely on a global secret.

## 7.2 State Size

One of the most important factors in terms of protocol practicality is the size of the state at each router that needs to be maintained. Assume that we wish to provide fault-tolerant forwarding between each source and destination pair in the network. Let  $N$  be the number of routers in the network,  $R$  be the maximum number of links incident on a router and  $k$  be the value used in the *AdjacentFault*( $k$ ) assumption. PERLMAN, HERZBERG, HSER, each require  $O(N^2)$  state, since each individual router maintains state for each (source, destination) pair. WATCHERS reduces the state requirement to  $O(RN)$ , where each individual router keeps seven counters for each of its neighbors for each destination. In SecTrace, if a router monitors paths to all destinations, then  $O(N)$  space is required (assuming that there is only one path in active use between a given source and destination).

Our protocols require a router  $r$  to record state for each path-segment  $P_r$  that it monitors. By construction, this is  $O(k \times R^{k+1})$  for Protocol  $\Pi_2$  and  $O(\min\{R^{k+1}, N\})$  for Protocol  $\Pi_{k+2}$ . In practice, though, we expect  $|P_r|$  to be much

---

<sup>1</sup>DES and AES are two well known encryption algorithms providing confidentiality.

smaller as noted in Chapter 5.

### 7.3 Synchronization

Protocols that validate aggregate traffic require synchronization in order to agree on a time interval during which to collect traffic information. WATCHERS synchronizes all routers using a snapshot algorithm [24]. In `Protocol  $\Pi_2$` , for each path-segment  $\pi$  in  $P_r$ , a router  $r$  synchronizes with all the routers in  $\pi$  to agree on when and for how long the next measurement interval  $\tau$  will be. It would probably be more efficient, though, to have all of the routers in the network synchronize with each other instead of having many more, smaller synchronization rounds. Perfect synchronization would not be necessary in practice, since the traffic validation function  $TV$  could be written to accommodate a small skew. The synchronization requirements for `Protocol  $\Pi_{k+2}$`  are lower than for `Protocol  $\Pi_2$` . In `Protocol  $\Pi_{k+2}$` , for each path-segment  $\pi$  that a router  $r$  monitors,  $r$  needs to agree with only the other end router  $r'$  of  $\pi$ . This is the same requirement as that for `SecTrace`.

Satellite receivers [15] can provide very accurate clock synchronization. For example, commercially available Global Positioning System receivers typically provide an accuracy within 50 nanoseconds to 1 millisecond referenced to UTC time, while Geostationary Operational Environment Satellite receivers provide an accuracy within 100 microseconds. Another possibility is to use clock-synchronization protocols, such as NTP [80], SNTP [81]. These protocols can provide accuracy on the order of milliseconds over the Internet, and can achieve accuracy within 200 microseconds in local area networks.

## 7.4 Issues

### 7.4.1 Multipaths

Routing protocols can take advantage of multiple paths with equal cost for load balancing purposes. This can be problematic for some of the Byzantine detection protocols<sup>2</sup>, since these protocols assume some knowledge of the path that a packet will take, at least in the stable state. For these protocols, the special case is that of multiple paths with the same administrative cost to a destination. For example, assume there exist two paths between routers  $a$  and  $e$ :  $\langle a, b, c, e \rangle$  and  $\langle a, b, d, e \rangle$ . If  $a$  monitors 3-path-segments then for a packet destined to  $e$ ,  $a$  needs to know whether  $b$  forwards that packet to  $c$  or  $d$ , so that  $a$  can keep the fingerprint of that packet for the corresponding 3-path-segment, either  $\langle a, b, c \rangle$  or  $\langle a, b, d \rangle$ .

Fortunately, the current generation of routers uses a deterministic hash algorithm to spread the traffic load across available interfaces (e.g., Cisco Express Forwarding [29] and Juniper routers with an Internet Processor ASIC [61]). Thus, a router can predict the path that a packet will take in the stable state based on its own routing tables and the hash functions.

### 7.4.2 TTL

One technical difficulty with conservation of content is that packets are naturally modified as they traverse routers. In particular, the TTL field in the IP header is decremented and the IP header checksum is updated.

Protocols that compute fingerprints based on only immutable content of the packets are vulnerable to attacks changing TTL field of the packets.

Protocol  $\Pi_{k+2}$ , Protocol  $\Pi_2$ , and Protocol  $\chi$  address this issue by having each router compute fingerprints based on the TTL value and checksum at one end of the path  $\pi$ .

---

<sup>2</sup>For Byzantine detection protocols that do not rely on source routing but rely on dynamic routing such as link state routing.

Another option is to keep these fields as a part of a fingerprint along with a one-way hash value over the immutable content.

### **7.4.3 Multicast**

Multicast forwards traffic along more than one path. Since conservation of flow as stated by WATCHERS inherently assumes single-path communication, WATCHERS can not be easily extended for multicast communications. All of the other protocols discussed above, including our own, can be extended easily for multicast. Doing so would require the routers to agree on the network topology and the multicast tree (something that, for example with MOSPF [89] would be straightforward to do).

### **7.4.4 Fragmentation**

Fragmentation does not change the data that is carried in network traffic, but rather changes the way that data is packetized. Thus, traffic validation based on packets is sensitive to fragmentation, while traffic validation based on data is not. Of the protocols mentioned in Chapter 3, only WATCHERS bases its traffic validation just on the data (it uses byte count implementing a traffic validation mechanism based on conservation of flow traffic policy) and is therefore insensitive to fragmentation. However, fragmentation occurs very rarely in the Internet [116], and so we don't consider sensitivity to fragmentation to be a practical concern.

Some protocols rely on source routing, such as PERLMAN, HERZBERG and HSER. Embedding the source routing information into a packet increases the packet size. Consequently this might lead to a severe drawback: fragmentation, in which case the MAC values are no longer valid for the fragments. Furthermore, in HSER, the source router computes a MAC for each intermediate router along the path, and embeds these MACs and a sequence number into the packet as well as the source routing information. This increases the chance of fragmentation.

## **Acknowledgement**

Parts of Chapter 7 are reprints of the material as it appears in the IEEE Transactions on Dependable and Secure Computing, 2006, by Alper Tugay Mizrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage.

## Chapter 8

# Conclusion

Network routers occupy a unique role in modern distributed systems. They are responsible for cooperatively shuttling packets amongst themselves in order to provide the *illusion* of a network with universal point-to-point connectivity. However, this illusion is shattered – as are implicit assumptions of availability, confidentiality or integrity – when network routers are subverted to act in a malicious fashion. By manipulating, diverting or dropping packets arriving at a compromised router, an attacker can trivially mount denial-of-service, surveillance or man-in-the-middle attacks on end host systems. Consequently, Internet routers have become a choice target for would-be attackers and thousands have been subverted to these ends.

In this dissertation, first, we study the problem space and describe a general framework for understanding the literature on detecting malicious routers via packet forwarding behavior. We describe how traffic validation is the basis for all such schemes and explore the design space of protocols that implement such a detector. Next, we formally specify this problem of detecting routers with incorrect packet forwarding behavior.

In Chapter 5, we further present two concrete protocols that differ in accuracy, completeness, and overhead – one of which is likely inexpensive enough for practical implementation at scale. `Protocol  $\Pi_2$`  is based on traffic validation per path-segment nodes and is strong-complete and accurate with precision of 2. `Protocol  $\Pi_{k+2}$`  is

based on traffic validation per path-segment ends and is strong-complete and accurate with precision of  $k$ , where  $k$  is the maximum number of adjacent faulty routers. Next, we present a prototype system, called *Fatih*, that implements this approach on a PC router and describe our experiences with it. We show that *Fatih* is able to detect and isolate a range of malicious router actions with acceptable overhead and complexity.

To the best of our knowledge, `Protocol  $\chi$`  in Chapter 6 is the first serious attempt to distinguish between a router dropping packets maliciously and a router dropping packets due to congestion. Previous work, including `Protocol  $\Pi_2$`  and `Protocol  $\Pi_{k+2}$` , has approached this issue using a static user-defined threshold, which is fundamentally limiting. `Protocol  $\chi$`  dynamically infers, based on measured traffic rates and buffer sizes, the number of congestive packet losses that will occur. Once the ambiguity from congestion is removed, subsequent packet losses can be attributed to malicious actions. Because of nondeterminism introduced by imperfectly synchronized clocks and scheduling delays, `Protocol  $\chi$`  still uses user-defined thresholds, but these thresholds are independent of the properties of the traffic. Hence, this approach does not suffer from the limitations of static thresholds. We evaluated the effectiveness of `Protocol  $\chi$`  through an implementation and deployment in a small network. We show that even fine grained attacks, such as stopping a host from opening a connection by discarding the SYN packet, are detected.

We believe our work is an important step in being able to tolerate attacks on key network infrastructure components.

In a short time, there have been significant advances in this domain including our research. While none of these protocols has yet been deployed in a production network, they are quickly becoming cheap enough and precise enough to be a viable option against router-oriented attacks. Future work should focus on practical implementation and optimization of these failure detectors in real routers.

## Appendix A

# Set Reconciliation Algorithm

The most promising way to implement the traffic validation function for conservation of content is based on "set reconciliation". In [82], where Minsky and Trachtenberg also present two instances of the problem: one for client-server model and one for general peer-to-peer model, the problem is defined as the following:

Consider a pair of hosts  $A$  and  $B$  that each have a database of  $b$ -bit vectors, stored in sets  $S_A$  and  $S_B$  respectively. The goal is to learn  $\delta_A = S_B \setminus S_A$  and  $\delta_B = S_A \setminus S_B$ .

Later on, Minsky et al. improve these algorithms in [84, 83].

In [84], the algorithm, which also works for multisets, computes a "synopsis" based on a recovery bound parameter  $\bar{m}$ , and a redundancy factor  $k$ .  $\bar{m}$  is an estimation of the sizes of  $\delta_A$  and  $\delta_B$ . A synopsis is an encoding of a set as polynomials using a *characteristic polynomial*  $X_S(z)$  of a set  $S = \{s_1, s_2, \dots, s_n\}$  at point  $z$ :  $X_S(z) = (z - s_1)(z - s_2)\dots(z - s_n)$ .

Their algorithm is as follows:

1. Both parties  $A$  and  $B$  agree on a set  $Z$  of  $\bar{m}$  points that do not overlap with the values in  $S_A$  and  $S_B$ . For each point  $z \in Z$ ,  $A$  and  $B$  compute the characteristic polynomial values  $X_{S_A}(z)$  and  $X_{S_B}(z)$ , respectively, and exchange these values.

The characteristic polynomials are also evaluated at  $k$  randomly chosen points in order to determine if the reconciliation is impossible due to underestimation.

2. Notice that, at point  $z \in Z$ ,  $\frac{X_{S_A}(z)}{X_{S_B}(z)} = \frac{X_{\delta_A}(z)}{X_{\delta_B}(z)}$ , because the factors for the elements in both  $A$  and  $B$  cancel out.

The values of  $\frac{X_{S_A}(z)}{X_{S_B}(z)}$  are interpolated to recover the coefficients of the reduced rational function  $\frac{X_{\delta_A}(z)}{X_{\delta_B}(z)}$ .

3. The elements of  $\delta_A$  and  $\delta_B$  can be recovered by factoring  $X_{\delta_A}(z)$  and  $X_{\delta_B}(z)$ .

Let  $m$  be the actual size of symmetric distance,  $m = |S_A \oplus S_B| = |\delta_A| + |\delta_B|$  and  $b$  be the number of bits used to encode each element in the sets. If  $m < \bar{m}$ , then the reconciliation succeeds. Otherwise, with a probability of  $\left(\frac{|S_A|+|S_B|}{2^b}\right)^k$ , the algorithm determines that  $m > \bar{m}$ , and returns *fail*. The size of the synopsis is  $(\bar{m} + k + 1)(b + 1) - 1$  bits, the time to compute the synopsis is  $O(|S|(\bar{m} + k)b)$ , and the time to reconcile the sets is  $O(b\bar{m}^3 + b\bar{m}k)$ . This algorithm provides one round communication but it is hard to estimate the recovery bound parameter  $\bar{m}$ .

Furthermore, the last step of their algorithm is redundant, in our case. As our goal is to determine the sizes of  $\delta_A$  and  $\delta_B$ , we do not need to factor the  $X_{\delta_A}(z)$  and  $X_{\delta_B}(z)$  polynomials. Their degree gives us what we are looking for.

In [83], Minsky and Trachtenberg improve these result significantly by adopting *divide-and-conquer* strategy. They partition the sets recursively, until reconciliation is successful for each partition. The algorithm uses the following parameters: a recovery bound  $\bar{m}$ , which is normally a small constant such as 1, 3, 5; a redundancy factor  $k$ , which, for set sizes  $2^{20}$  and  $b = 32$ ,  $k = 1$  gives an error probability of  $2^{-11}$  to detect underestimation; as well as a partition factor  $p$ , which is used to determine the number of partitions created at each recursion. They present a tree data structure to compute the characteristic polynomial efficiently.

They report the following results for the expected case:

- computational complexity for computing synopsis:

$$O(|S| b(\bar{m} + k) \log_p(|S| / \bar{m} + p))$$

- communication complexity for sending synopsis:  $O(mp b(\bar{m}^2 + k))$

- expected number of rounds for communication:  $O\left(\log_p\left(\frac{m}{\bar{m}+1}\right)\right)$
- computational complexity for reconciliation:  $O(mp b(\bar{m} + k))$

## Appendix B

# Properties of Protocol $\Pi_2$ and Protocol $\Pi_{k+2}$

### B.1 Basic Theorems

**Theorem 1.** *If a router  $r$  is traffic faulty at some time  $t$  and  $AdjacentFault(k)$  holds, then there exists a path-segment  $\pi$ , such that:*

- $r \in \pi$
- $r$  is traffic faulty in  $\pi$  during some  $\tau$  that contains  $t$
- only the first and last routers of  $\pi$  are correct
- $3 \leq |\pi| \leq k + 2$

*Proof.* If  $r$  is traffic faulty at time  $t$ , then there is a path  $\Pi$ , such that  $r$  is traffic faulty in  $\Pi$  during some  $\tau$  that contains  $t$ . From the system assumption, the source and sink routers of  $\Pi$  are correct, and so  $\Pi$  must contain at least three routers in order to include the faulty router  $r$ .

For each path-segment  $\pi$  of  $\Pi$  that contains  $r$ ,  $r$  is traffic faulty in  $\pi$  during  $\tau$ . Given  $AdjacentFault(k)$ ,  $r$  can be in a group of no fewer than one and no more than  $k$  adjacent faulty routers. This group, by definition, is bounded on both sides by correct routers. □

**Theorem 2.** *If, for a path-segment  $\pi$ ,  $TV(\pi, \text{info}(h, \pi, \tau), \text{info}(j, \pi, \tau))$  is false where  $1 \leq h < j \leq |\pi|$ , then there exists a link  $\langle i, i + 1 \rangle$  such that  $TV(\pi, \text{info}(i, \pi, \tau), \text{info}(i + 1, \pi, \tau))$  is false and  $h \leq i < i + 1 \leq j$ .*

*Proof.* By contradiction. Assume that there is no link  $\langle i, i + 1 \rangle$  such that  $TV(\pi, \text{info}(i, \pi, \tau), \text{info}(i + 1, \pi, \tau))$  is false and  $h \leq i < i + 1 \leq j$ . For each link  $\langle i, i + 1 \rangle$  such that  $h \leq i < i + 1 \leq j$ ,  $TV(\pi, \text{info}(i, \pi, \tau), \text{info}(i + 1, \pi, \tau))$  is true. Since  $TV$  is transitive,  $TV(\pi, \text{info}(h, \pi, \tau), \text{info}(j, \pi, \tau))$  is true, which leads us a contradiction.  $\square$

## B.2 Properties of Protocol $\Pi_2$

**Theorem 3.** *The Protocol  $\Pi_2$  is 2-Accurate.*

*Proof.* By construction, all suspicions are path-segments of length 2. For a correct router  $s$  to suspect  $(\pi, \tau)$ , that router must find  $TV(\pi, \text{info}(i, \pi, \tau), \text{info}(i + 1, \pi, \tau))$  to be false, for some  $\pi$  that contains  $i$  and  $i + 1$ . Furthermore, since the traffic information is digitally signed, the two routers did report this traffic information. Hence, at least one of the two routers must be traffic faulty or protocol faulty.  $\square$

**Theorem 4.** *The Protocol  $\Pi_2$  is 2-FC Complete.*

Formally, if a router  $r$  is traffic faulty at some time  $t$ , then all correct routers eventually suspect  $(\pi, \tau)$  for some path-segment  $\pi : |\pi| \leq 2$  and some  $\tau$  containing  $t$  such that there is a router  $r'$  that was faulty in  $\pi$  at time  $t'$  in  $\tau$  and is fault-connected to  $r$ .

*Proof.* By Theorem 1, if a router  $r$  is traffic faulty at time  $t$ , then there exists a path-segment  $\pi'$ , such that:  $r \in \pi'$ ,  $r$  is also traffic faulty in  $\pi'$  during  $\tau$  containing  $t$ , and only the first and last routers of  $\pi'$  (which we'll call  $f$  and  $\ell$ ) are correct and  $3 \leq |\pi'| \leq k + 2$ .

By construction of  $P_f$  and  $P_\ell$ , both  $f$  and  $\ell$  monitor at least one path-segment  $\pi''$  such that  $\{f, r, \ell\} \in \pi''$  and  $\pi''$  contains  $\pi'$ .

Both  $f$  and  $\ell$  compute  $TV(\pi'', \text{info}(f, \pi'', \tau), \text{info}(\ell, \pi'', \tau))$  to be false. By Theorem 2, there exists a 2-path-segment  $\pi = \langle i, i + 1 \rangle$  such that  $TV(\pi'', \text{info}(i, \pi'', \tau),$

$info(i + 1, \pi'', \tau)$  is false where  $f \leq i < i + 1 \leq \ell$ . Since all routers between  $f$  and  $\ell$  are faulty and fault-connected to  $r$ , at least one of  $\{i, i+1\}$  is faulty and fault-connected to  $r$ .

Both correct routers  $f$  and  $\ell$  detect this failure and reliably broadcast to all correct routers the evidence  $info(i, \pi'', \tau), info(i + 1, \pi'', \tau)$ , which are digitally signed by routers  $i$  and  $i + 1$ , respectively. Eventually all correct routers suspect  $\pi = \langle i, i + 1 \rangle$ .

□

### B.3 Properties of Protocol $\Pi_{k+2}$

**Theorem 5.** *The Protocol  $\Pi_{k+2}$  is  $(k+2)$ -Accurate.*

*Proof.* If a correct router suspects  $(\pi, \tau)$ , then  $|\pi| \leq a$  and some router  $r \in \pi$  was faulty in  $\pi$  during  $\tau$ .

For a correct router to suspect a path-segment  $\pi$ , router  $s$  that is either the first or last router of  $\pi$  must announce that ‘ $\pi$  is unreliable’.

1. If this announcement is incorrect, then  $s$  is protocol faulty.
2. If this announcement is correct, then  $s$  found  $TV(\pi, info(1, \pi, \tau), info(|\pi|, \pi, \tau))$  to be false. Assume there exists no faulty router in  $\pi$  exhibiting faulty behavior with respect to  $\pi$  during  $\tau$ . Then, each router in  $\pi$  forwards the traffic traversing  $\pi$  correctly. Since both router 1 and router  $s$  are correct, they collect and exchange traffic information correctly. Thus, both routers will find  $TV(\pi, info(1, \pi, \tau), info(|\pi|, \pi, \tau))$  to be true, which contradicts our assumption.

A correct router applies the Protocol  $\Pi_{k+2}$  to  $x$ -path-segments where  $x \leq k + 2$ . Hence, Protocol  $\Pi_{k+2}$  is  $(k+2)$ -Accurate. □

**Theorem 6.** *The Protocol  $\Pi_{k+2}$  is  $(k+2)$ -Complete.*

We show that if a router  $r$  is traffic faulty at some time  $t$ , then all correct routers eventually suspect  $(\pi, \tau)$  for some path-segment  $\pi : |\pi| \leq k + 2$  such that  $r$  was traffic faulty in  $\pi$  at  $t$ , and for some interval  $\tau$  containing  $t$ .

*Proof.* Let  $r$  have introduced discrepancy into the traffic passing through itself during  $\tau$  containing  $t$ . Then, from Theorem 1, there exists a path segment  $\pi$  such that:

- $r \in \pi$
- $r$  is traffic faulty in  $\pi$  during  $\tau$  containing  $t$
- only  $f$  and  $\ell$  — the first and last routers of  $\pi$  — are correct
- $3 \leq |\pi| \leq k + 2$

$f$  and  $\ell$  monitor  $\pi$  and apply the `PROTOCOL`  $\Pi_{k+2}$  for  $\pi$ . After exchanging their traffic information, both  $f$  and  $\ell$  compute  $TV(\pi, info(f, \pi, \tau), info(\ell, \pi, \tau))$  to be false and suspect  $\pi$  and disseminate this information to the all other correct routers by reliable broadcast. Since  $\pi$  contains a traffic faulty router  $r$  and the length of  $\pi$  may be at most  $k + 2$ , the `PROTOCOL`  $\Pi_{k+2}$  is  $(k+2)$ -Complete.  $\square$

## Acknowledgement

Appendix B is a reprint of the material as it appears in the IEEE Transactions on Dependable and Secure Computing, 2006, by Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage.

## Appendix C

# Properties of Protocol $\chi$

**Theorem 7.** *The Protocol  $\chi$  is accurate.*

*Proof.* When a correct router  $e''$  receives a suspicious link  $\ell = \langle e, e' \rangle$  announcement originated by router  $e$ ,  $e''$  detects  $\ell$  as faulty. Then there must be at least one faulty router in  $\ell$ :

- If  $e$  is *faulty*, and it announces its link  $\ell$  as faulty; indeed  $\ell$  has a faulty router:  $e \in \ell$ . A *protocol faulty* router can always announce its link  $\ell$  as faulty.
- If  $e$  is correct, and suspects its neighbor  $e'$  announcing its link  $\ell$  as faulty, then  $e'$  must be faulty. We show this by considering each detection in Section. 6.2.2.
  - D-1a: Assume  $e'$  is correct, and it sends its traffic information  $Tinfo(r', Q_{in}, \langle e', e, r_d \rangle, \tau)$  to router  $e$  at the end of validation time interval  $\tau$ . The message must be delivered to  $e$  in  $\Delta$  time, which is a contradiction of the fact that  $e$  is correct yet does not receive this message.
  - D-1b: Assume  $e'$  is correct and sends digitally signed traffic information which is consistent and valid. Correct router  $e$  validates the signature and the consistency of the traffic information. This contradicts the fact that  $e$  suspects  $e'$ .
  - D-2a: Assume  $e'$  is correct. Then one of the following is true: 1)  $e'$  received traffic information from  $r_{s^*}$  in  $\Delta$  time, verified it and forwarded it to  $e$  in

the next  $\Delta$  time. This contradicts the fact that a correct router  $e$  did not receive the message. 2)  $e'$  did not verify traffic information from  $r_{s^*}$  or did not receive the message in  $\Delta$  time. Then it should have detected  $\ell = \langle r_{s^*}, e' \rangle$  and announced the detection. This contradicts the fact that correct router  $e$  did not receive the detection announcement.

- D-2b: Assume  $e'$  is correct, and forwards traffic information to  $e$  only if it validates the signature. Then the correct router  $e$  validates the signature. This contradicts the failure of the digital signature verification.
- D-2c: Assume  $e'$  is correct, and forwards traffic correctly. Since  $e'$  is correct, all traffic information of  $S$ , which  $e'$  sent to  $e$ , is verified by  $e'$ . With the input of  $S$  verified by correct router  $e'$  and the input of  $D$  collected by  $e$ ,  $TV$  predicate evaluates to *true*. This contradicts the fact that  $TV$  evaluated to *false*.

All detections by `Protocol  $\chi$`  are 2-path-segments. Hence, it is 2-accurate.

□

**Theorem 8.** *The Protocol  $\chi$  is complete.*

*Proof.* If a router  $e$  is traffic faulty<sup>1</sup> at some time  $t$ , then all correct routers eventually suspect  $(\ell, \tau)$  for some link  $\ell$  such that  $e \in \ell$  and  $e$  was traffic faulty at  $t$ , and for some interval  $\tau$  containing  $t$ .

Let  $e$  have dropped packets maliciously from the traffic passing through itself towards  $e'$  during  $\tau$  containing  $t$ . At the end of traffic validation round  $\tau$ ,  $e'$  will validate  $Q$  associated with  $\ell$ .

As we assume that adjacent routers cannot be compromised in our threat model, all neighbors of  $e$  are correct and collect traffic information appropriately during  $\tau$ . At the end of  $\tau$ , they send this information to  $e$  to forward to  $e'$ .

---

<sup>1</sup>As in [87], the protocol we develop assumes that the terminal routers are correct. This assumption is common to all detection protocols. The argument is based on *fate sharing*. If a terminal router is compromised then there is no way to determine what traffic was injected or delivered in the system. Thus, a compromised terminal router can always invisibly disrupt traffic sourced or sinked to its network. One could place terminal routers on each workstation thus limiting the damage they can wreak to only a workstation.

Then one of the following is true:

- D-2c:  $e$  passes this information to  $e'$ . The complete  $TV$  evaluates to *false* with these correct inputs. So  $e'$  detects  $\ell = \langle e, e' \rangle$ , where  $e \in \ell$ .
- D-2b:  $e$  passes this information to  $e'$  after tampering with the content in order to hide the attack.  $e'$  fails to verify the signatures, so  $e'$  detects  $\ell = \langle e, e' \rangle$ , where  $e \in \ell$ .
- D-2a:  $e$  passes its own copy of traffic information to  $e'$  to hide the attack. Then  $e'$  expects  $e$  to detect  $r_{s^*}$  whose traffic information has not been forwarded to  $e'$ .  
 1) If  $e$  detects  $\ell = \langle r_{s^*}, e \rangle$ , then  $e \in \ell$ . 1) If  $e$  fails to detect  $\ell = \langle r_{s^*}, e \rangle$ , then  $e'$  detects  $\ell = \langle e, e' \rangle$ , where  $e \in \ell$ .
- D-2a:  $e$  does not send any traffic information in  $S$  to  $e'$ . Due to the timeout mechanism, after  $2\Delta$  time,  $e'$  detects  $\ell = \langle e, e' \rangle$ , where  $e \in \ell$ .

□

## Acknowledgement

Appendix C is a reprint of the material as it appears in UCSD Technical Report, CS2007-0889, 2007, by Alper Tugay Mizrak, Keith Marzullo and Stefan Savage.

# Bibliography

- [1] Abilene Network. <http://abilene.internet2.edu/>, 2005.
- [2] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way packet loss metric for IPPM. *RFC 2680, IETF*, Sept. 1999.
- [3] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of tcp/ip with stationary random losses. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 231–242, New York, NY, USA, 2000. ACM Press.
- [4] X. Ao. Report on DIMACS Workshop on Large-Scale Internet Attacks, Sept. 2003.
- [5] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 281–292, New York, NY, USA, 2004. ACM Press.
- [6] K. Argyraki and D. R. Cheriton. Loose source routing as a mechanism for traffic policies. In *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 57–64, New York, NY, USA, 2004. ACM Press.
- [7] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing packet obituaries. In *Proceedings of ACM SIGCOMM HotNets-III*, 2004.
- [8] K. Arvind. Probabilistic clock synchronization in distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 5(5):474–487, 1994.
- [9] S. Athuraliya, S. Low, V. Li, and Q. Yin. REM: Active queue management. *IEEE Network*, 15(3):48–53, 2001.
- [10] I. Avramopoulos, H. Kobayashi, A. Krishnamurthy, and R. Wang. *Secure Routing*, chapter in *Network Security: Current Status and Future Directions*, C. Douligieris and D. N. Serpanos (editors). Wiley-IEEE Press, 2007.

- [11] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. In *Proceedings of INFOCOM 2004 Conference*, March 2004.
- [12] I. Avramopoulos and J. Rexford. Stealth Probing: Efficient Data-Plane Security for IP Routing. In *Proc. USENIX Annual Technical Conference*, May-Jun 2006.
- [13] A. Barbir, S. Murphy, and Y. Yang. Generic Threats to Routing Protocols, RFC 4593, Oct 2006.
- [14] C. N.-R. Baruch Awerbuch, David Holmer and H. Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *ACM Workshop on Wireless Security (WiSe)*, September 2002.
- [15] K. Behrendt and K. Fodero. The perfect time: An examination of time-synchronization techniques. In *DistribuTECH*, 2006.
- [16] J. Bellardo and S. Savage. Measuring packet reordering. In *ACM SIGCOMM Internet Measurement Workshop(IMW02)*, pages 97–105, 2002.
- [17] M. Bellare, R. Canetti, and H. Krawczyk. Message authentication using hash functions: the HMAC construction. *CryptoBytes*, 2(1):12–15, Spring 1996.
- [18] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking (TON)*, 7(6):789–798, 1999.
- [19] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. *Lec. Notes in CS*, 1666:216–233, 1999.
- [20] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Comm. of the ACM*, 13(7):422–426, July '70.
- [21] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 115–124, May 1998.
- [22] N. Cardwell, S. Savage, and T. E. Anderson. Modeling tcp latency. In *INFOCOM*, pages 1742–1751, 2000.
- [23] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
- [24] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.
- [25] H.-Y. Chang, S. F. Wu, and Y. F. Jou. Real-time protocol analysis for detecting link-state routing protocol attacks. *ACM Trans. Inf. Syst. Secur.*, 4(1):1–36, 2001.

- [26] S. Cheung. An efficient message authentication scheme for link state routing. In *ACSAC*, pages 90–98, 1997.
- [27] S. Cheung and K. Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *New Security Paradigms Workshop*, 1997.
- [28] Cisco Systems. Detecting and Analyzing Network Threats With NetFlow.
- [29] Cisco Systems. Load balancing with cisco express forwarding.
- [30] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [31] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 9(3):280–292, 2001.
- [32] Emulab - Network Emulation Testbed. <http://www.emulab.net>, 2006.
- [33] W. Feghali, B. Burres, G. Wolrich, and D. Carrigan. Security: Adding protection to the network via the network processor. *Intel Technology Journal*, 06:40–49, Aug. 2002.
- [34] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.
- [35] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.
- [36] T. Garfinkel. Traps and pitfalls: Practical problems in system call interposition based security tools. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.
- [37] Gaus. Things to do in Cicoland when you’re dead, Jan. 2000. [www.phrack.org](http://www.phrack.org).
- [38] GNU Zebra. <http://www.zebra.org>.
- [39] S. Goldberg, D. Xiao, B. Barak, and J. Rexford. Measuring path quality in the presence of adversaries: The role of cryptography in network accountability. Technical report, Princeton University, 2007.
- [40] O. Goldreich. *Foundations of Cryptography*, volume Basic Tools. Cambridge University Press, 2001.
- [41] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin. Working around BGP: An incremental approach to improving security and accuracy in interdomain routing. In *NDSS ’03: Proceedings of the 2003 Symposium on Network and Distributed System Security*, Washington, DC, USA, 2003. IEEE Computer Society.

- [42] M. T. Goodrich. Efficient and secure network routing algorithms, Jan 2001. Provisional patent filing.
- [43] T. J. Hacker, B. D. Noble, and B. D. Athey. The effects of systemic packet loss on aggregate tcp flows. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–15, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.
- [44] D. Harkins and D. Carrel. The Internet Key Exchange (IKE), RFC 2409, IETF, Nov. 1998.
- [45] R. Hauser, T. Przygienda, and G. Tsudik. Reducing the cost of security in link-state routing. In *NDSS '97: Proceedings of the 1997 Symposium on Network and Distributed System Security*, page 93, Washington, DC, USA, 1997. IEEE Computer Society.
- [46] R. Hauser, T. Przygienda, and G. Tsudik. Lowering security overhead in link state routing. *Computer Networks*, 31(9):885–894, 1999.
- [47] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS), RFC 1930, IETF, Mar. 1996.
- [48] L. He. Recent developments in securing internet routing protocols. *BT Technology Journal*, 24(4):180–196, 2006.
- [49] A. Herzberg and S. Kuten. Early detection of message forwarding faults. *SIAM J. Comput.*, 30(4):1169–1196, 2000.
- [50] C. V. Hollot, V. Misra, D. F. Towsley, and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proc. of the INFOCOM '01*, pages 1726–1734, apr 2001.
- [51] K. J. Houle, G. M. Weaver, N. Long, and R. Thomas. Trends in denial of service attack technology. CERT Coordination Center Technical Report, Oct. 2001.
- [52] Y. Hu, A. Perrig, and D. Johnson. Efficient security mechanisms for routing protocols. In *NDSS '03: Proceedings of the 2003 Symposium on Network and Distributed System Security*, Washington, DC, USA, 2003. IEEE Computer Society.
- [53] Y.-C. Hu, D. B. Johnson, and A. Perrig. SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks. *Ad Hoc Networks*, 1(1):175–192, 2003.
- [54] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: secure path vector routing for securing BGP. *ACM SIGCOMM Computer Communication Review*, 34(4):179–192, 2004.

- [55] D. Huang, A. Sinha, and D. Medhi. A Double Authentication Scheme To Detect Impersonation Attack In Link State Routing Protocols. *IEEE International Conference on Communications (ICC)*, May 2003.
- [56] J. R. Hughes, T. Aura, and M. Bishop. Using conservation of flow as a security mechanism in network protocols. In *IEEE Symp. on Security and Privacy*, pages 132–131, 2000.
- [57] V. Jacobson. The Traceroute Manual Page, Lawrence Berkeley Laboratory, Dec. 1988.
- [58] K. Jain and R. Sekar. User-level infrastructure for system call interposition: A platform for intrusion detection and confinement. In *Proc. Network and Distributed Systems Security Symposium*, pages 19–34, 2000.
- [59] W. Jiang and H. Schulzrinne. Modeling of packet loss and delay and their effect on real-time multimedia service quality. In *Proc. NOSSDAV*, 2000.
- [60] Y. Jou, F. Gong, C. Sargor, X. Wu, S. Wu, H. Chang, and F. Wang. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00 Proceedings*, volume 2, pages 69–83, 2000.
- [61] Juniper Networks. JUNOS 6.4 Routing Protocols Configuration Guide.
- [62] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private communication in a public world*, volume Second edition. Prentice Hall, 2002.
- [63] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP), RFC 2406, IETF, Nov. 1998.
- [64] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol, RFC 2401, IETF, Nov. 1998.
- [65] S. Kent, C. Lynn, J. Mikkelsen, and K. Seo. Secure Border Gateway Protocol (Secure-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, Apr. 2000.
- [66] S. T. Kent. Securing the border gateway protocol: A status update. In *Seventh IFIP TC-6 TC-11 Conference on Communications and Multimedia Security*, Oct 2003.
- [67] A. P. Kosoresow and S. A. Hofmeyr. Intrusion detection via system call traces. *IEEE Software*, 14(5):35–42, 1997.
- [68] A. Kuzmanovic. The power of explicit congestion notification. In *Proc. of the SIGCOMM '05*, pages 61–72, 2005.

- [69] A. Kuzmanovic and E. W. Knightly. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, New York, NY, USA, 2003. ACM Press.
- [70] C. Labovitz, A. Ahuja, and M. Bailey. Shining light on dark address space, Nov. 2001. Arbor Networks Tech. Rep.
- [71] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, Nov 1981.
- [72] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. on Programming Languages and Systems*, 4(3):382–401, 1982.
- [73] R. J. Larsen and M. L. Marx. *Introduction to Mathematical Statistics and its Application; 4 edition*. Prentice Hall, 2005.
- [74] L. Le, J. Aikat, K. Jeffay, and F. D. Smith. The effects of active queue management on web performance. In *Proc. of the SIGCOMM '03*, pages 265–276, 2003.
- [75] S. Lee, T. Wong, and H. S. Kim. Secure split assignment trajectory sampling: A malicious router detection system. *dsn*, 0:333–342, 2006.
- [76] M. G. Luby and L. Michael. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, Princeton, NJ, USA, 1994.
- [77] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, 1997.
- [78] G. Mathur, V. N. Padmanabhan, and D. Simon. Securing routing in open networks using secure traceroute. Technical Report MSR-TR-2004-66, Microsoft Research Technical Report, July 2004.
- [79] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the internet. *SIGCOMM Comput. Commun. Rev.*, 35(2):37–52, 2005.
- [80] D. L. Mills. Network time protocol (version 3) specification, implementation. *RFC 1305, IETF*, Mar. 1992.
- [81] D. L. Mills. Simple network time protocol (sntp) version 4 for ipv4, ipv6 and osi. *RFC 4330, IETF*, Jan. 2006.
- [82] Y. Minsky and A. Trachtenberg. Efficient reconciliation of unordered databases. Technical Report TR1999-1778, Cornell University, 1999.
- [83] Y. Minsky and A. Trachtenberg. Practical set reconciliation. Technical Report Technical Report 2002-03, Boston University, 2002.

- [84] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. In *Int. Symp. on Information Theory*, page 232, June 2001.
- [85] V. Mittal and G. Vigna. Sensor-based intrusion detection for intra-domain distance-vector routing. In *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 127–137, New York, NY, USA, 2002. ACM Press.
- [86] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and isolating malicious routers. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 538–547, 2005.
- [87] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Detecting and isolating malicious routers. *IEEE Transactions on Dependable and Secure Computing*, 3(3):230–244, Jul-Sep 2006.
- [88] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. Packet reordering metric for IPPM, Mar. 2003.
- [89] J. T. Moy. Multicast Extensions to OSPF, RFC 1584, IETF, Mar. 1994.
- [90] J. T. Moy. OSPF Version 2, RFC 2328, IETF, Apr. 1998.
- [91] S. Murphy, O. Gudmundsson, R. Mundy, and B. Wellington. Retrofitting security into internet infrastructure protocols. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00 Proceedings*, volume 1, pages 3–17, 2000.
- [92] S. L. Murphy and M. R. Badger. Digital signature protection of the ospf routing protocol. In *SNDSS '96: Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96)*, page 93, Washington, DC, USA, 1996. IEEE Computer Society.
- [93] National Institute of Standards and Technology (NIST). Digital signature standard. *FIPS PUBS 186*, May 1994.
- [94] National Institute of Standards and Technology (NIST). Secure hash standard. *FIPS PUBS 180-1*, Apr. 1995.
- [95] National Institute of Standards and Technology (NIST). Data encryption standard. *FIPS PUBS 46-3*, Oct. 1999.
- [96] National Institute of Standards and Technology (NIST). Advanced encryption standard. *FIPS PUBS 197*, Nov. 2001.
- [97] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: a simple model and its empirical validation. In *SIGCOMM '98: Proceedings of*

the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, pages 303–314, New York, NY, USA, 1998. ACM Press.

- [98] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Computer Communications Review*, 33(1):77–82, 2003.
- [99] P. Papadimitratos and Z. J. Haas. Securing the Internet routing infrastructure. *Communications Magazine, IEEE*, 40(10):60–68, 2002.
- [100] P. Papadimitratos and Z. J. Haas. Secure link state routing for mobile ad hoc networks. In *SAINT-W '03: Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*, page 379, Washington, DC, USA, 2003. IEEE Computer Society.
- [101] C. Partridge, A. C. Snoeren, W. T. Strayer, B. Schwartz, M. Condell, and I. Castineyra. Fire: flexible intra-as routing environment. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 19(3), mar 2001.
- [102] D. Pei, D. Massey, and L. Zhang. Detection of invalid routing announcements in the RIP protocol. In *IEEE Global Communications Conference (Globecom'03)*, volume 3, pages 1450–1455, dec 2003.
- [103] K. Pentikousis and H. Badr. Quantifying the deployment of TCP options - A comparative study. *Communications Letters, IEEE*, 8(10):647–649, 2004.
- [104] R. Perlman. *Network Layer Protocols with Byzantine Robustness*. PhD thesis, MIT LCS TR-429, Oct. 1988.
- [105] R. Perlman. *Interconnections: Bridges and Routers*. Addison Wesley Longman Publishing Co. Inc., 1992.
- [106] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Proc. Network and Distributed System Security Symposium, SNDSS '01*, Feb 2001.
- [107] D. Pullin, A. Corlett, B. Mandeville, and S. Critchley. Packet reordering: The minimal longest ascending subsequence metric, Feb. 2002.
- [108] D. Qu, B. Vetter, F. Wang, R. Narayan, S. Wu, Y. Jou, F. Gong, and C. Sargor. Statistical anomaly detection for link-state routing protocols. In *ICNP '98: Proceedings of the Sixth International Conference on Network Protocols*, page 62, Washington, DC, USA, 1998. IEEE Computer Society.
- [109] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. *RFC 3168, IETF*, 2001.
- [110] R. Rivest. The md5 message-digest algorithm, rfc 1321, 1992.

- [111] P. Rogaway. UMAC Performance (more).
- [112] L. A. Sanchez, W. C. Milliken, A. C. Snoeren, F. Tchakountio, C. E. Jones, S. T. Kent, C. Partridge, and W. T. Strayer. Hardware support for a hash-based ip traceback. In *2. DARPA Information Survivability Conference and Exposition (DISCEX II)*, pages 146–152, 2001.
- [113] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer. A secure routing protocol for ad hoc networks. In *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*, pages 78–89, Washington, DC, USA, 2002. IEEE Computer Society.
- [114] N. Shah. Understanding network processors. Master's thesis, University of California, Berkeley, September 2001.
- [115] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A case study of ospf behavior in a large enterprise network. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 217–230, New York, NY, USA, 2002. ACM Press.
- [116] C. Shannon, D. Moore, and K. C. Claffy. Beyond folklore: observations on fragmented traffic. *IEEE/ACM Trans. Netw.*, 10(6):709–720, 2002.
- [117] B. R. Smith and J. J. Garcia-Luna-Aceves. Efficient security mechanisms for the border gateway routing protocol. *Computer Communications*, 21(3):203–210, 1998.
- [118] B. R. Smith, S. Murthy, and J. J. Garcia-Luna-Aceves. Securing distance-vector routing protocols. In *SNDSS '97: Proceedings of the 1997 Symposium on Network and Distributed System Security*, page 85, Washington, DC, USA, 1997. IEEE Computer Society.
- [119] A. C. Snoeren and B. Raghavan. Decoupling policy from mechanism in internet routing. *SIGCOMM Comput. Commun. Rev.*, 34(1):81–86, 2004.
- [120] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 133–145. ACM Press, 2002.
- [121] I. Stamouli, P. G. Argyroudis, and H. Tewari. Real-time intrusion detection for ad hoc networks. In *WOWMOM '05: Proceedings of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM'05)*, pages 374–380, Washington, DC, USA, 2005. IEEE Computer Society.
- [122] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz. Listen and whisper: Security mechanisms for BGP. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

- [123] D. Taylor. Using a compromised router to capture network traffic, July 2002. Unpublished Technical Report.
- [124] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. In search of path diversity in ISP networks. In *Proc. of the ACM/SIGCOMM IMC*, pages 313–318, 2003.
- [125] The ns-2 network simulator. <http://www.isi.edu/nsnam/ns/>, 2005.
- [126] R. Thomas. ISP Security BOF, NANOG 28, June 2003.
- [127] User Mode Linux. <http://user-mode-linux.sourceforge.net/>.
- [128] D. Walton, A. Retana, and E. Chen. Advertisement of Multiple Paths in BGP, Internet Draft, Network Working Group, draft-walton-bgp-add-paths-05.txt, Aug. 2006.
- [129] T. Wan, E. Kranakis, and P. C. van Oorschot. S-RIP: A Secure Distance Vector Routing Protocol. In *ACNS*, pages 103–119, 2004.
- [130] D. Watson, F. Jahanian, and C. Labovitz. Experiences with monitoring ospf on a regional service provider network. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 204, Washington, DC, USA, 2003. IEEE Computer Society.
- [131] H. F. Wedde, J. A. Lind, and G. Segbert. Achieving Internal Synchronization Accuracy of 30 ms Under Message Delays Varying More Than 3 msec. In *WRTP99, 24th IFAC/IFIP Workshop on Real Time Programming*, 1999.
- [132] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford. Don't Secure Routing Protocols, Secure Data Delivery. In *Proc. 5th ACM Workshop on Hot Topics in Networks (Hotnets-V)*, Irvine, CA, Nov. 2006.
- [133] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the OSDI*, pages 255–270, Dec. 2002.
- [134] R. White. Securing BGP Through Secure Origin BGP. *The Internet Protocol Journal*, 6(3):15–22, Sep 2003.
- [135] S. F. Wu, F. yi Wang, B. M. Vetter, R. Cleaveland, Y. F. Jou, F. Gong, and C. Sargor. Intrusion detection for link-state routing protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, 1997.
- [136] M. Yajnik, S. B. Moon, J. F. Kurose, and D. F. Towsley. Measurement and modeling of the temporal dependence in packet loss. In *INFOCOM*, pages 345–352, 1999.
- [137] H. Yang, H. Y. Luo, F. Ye, S. W. Lu, and L. Zhang. Security in mobile ad hoc networks: Challenges and solutions. *IEEE Wireless Communications*, 11(1):38–47, 2004.

- [138] X. Yang. Nira: a new internet routing architecture. In *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 301–312, New York, NY, USA, 2003. ACM Press.
- [139] M. G. Zapata and N. Asokan. Securing ad hoc routing protocols. In *WiSE '02: Proceedings of the 3rd ACM workshop on Wireless security*, pages 1–10, New York, NY, USA, 2002. ACM Press.
- [140] K. Zhang. Efficient protocols for signing routing messages. In *Symposium on Network and Distributed Systems Security (NDSS '98)*, San Diego, California, 1998. Internet Society.
- [141] W. Zhang, R. Rao, G. Cao, and G. Kesidis. Secure routing in ad hoc networks and a related intrusion detection problem. In *IEEE Military Communications Conference*, 2003.
- [142] M. Zhao, S. W. Smith, and D. M. Nicol. Aggregated path authentication for efficient BGP security. In *CCS '05: Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 128–138, New York, NY, USA, 2005. ACM Press.
- [143] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Detection of invalid routing announcement in the internet. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 59–68, Washington, DC, USA, 2002. IEEE Computer Society.