# Detecting Disruptive Routers:
# A Distributed Network Monitoring Approach

*Kirk Alton Bradley*

**Department of Computer Science**

**University of California, Davis, CA 95616, USA**

**September 1997**

# Detecting Disruptive Routers:
# A Distributed Network Monitoring A p proach

By

KIRK ALTON BRADLEY
B.A. (University of Redlands) 1995

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

________________________________________ *Dr. Biswanath Mukherjee, Co-Chair*

________________________________________ *Dr. Ronald A. Olsson, Co-Chair*

________________________________________ *Dr. Matt Bishop*

Committee in Charge

1997

# Acknowledgments

I would like to formally thank Biswanath Mukherjee and Ronald Olsson. Their strong guidance and high expectations positively influenced every aspect of my research. I could not have asked for more diligence and care than they have shown during the writing of my thesis.

I would like to thank Matt Bishop whose helpful comments and suggestions have improved the quality of my thesis.

I would also like to extend a special thanks to Steven Cheung. My discussions with Steven influenced much of my research. His advice has always been thoughtful and sound.

Thank you to my fellow Master of Science, James Pace. His technical advice and sense of humor made my research a more enjoyable experience.

Nick Puketza has been a source of both technical knowledge and genuine good heartedness. Thank you, Nick.

Thanks to Chris Wee, for his helpful meetings, advice, and suggestions.

I would like to thank my friends and colleagues from UC Davis: Fritz Barnes, John Black, Steven Cheung, Ted Krovetz, Eric La Mar, Julie Lang, Dave O'Brien, James Pace, Nick Puketza, Steve Samorodin, and Scott Walnum.

I would like to acknowledge the National Security Agency (NSA) for supporting this research under grant number: DoD MDA904-96-1-0118.

Finally, I would like to thank my wife Bridget. A great undertaking like a thesis can be hard on a spouse, but Bridget has always been the perfect blend of empathetic and encouraging.

Kirk Alton Bradley

# Contents

# Chapter 1

# Introduction

## 1.1 Overview

### 1.1.1 Motivation

Internetworking is the interconnection of heterogeneous and autonomous computer systems, and is the core concept in today's Internet. The over four million computers that make up this global network span many boundaries between government, business, education, and private sector autonomous systems. An *autonomous system* (AS) is a set of routers and networks controlled by one administrative authority [Come95]. A *router* is a special purpose computer which, when it receives a communication transmission (*packet*) from one of its incoming interfaces, retransmits that packet on an outgoing interface in order to get the packet closer to its destination [Come95]. When computers need to communicate, each of their packet transmissions often traverses one or more routers, which may belong to one or more AS. It is the responsibility of each AS to protect the traffic traversing its routers against security breaches. Securing networks is a hard problem. We propose a protocol to improve the security of an AS by detecting routers that drop or misroute packets.

### 1.1.2 Our Model and Assoc iated Terminology

We model an AS as a directed graph, in which each node represents a router and each edge represents a communication *link* or *interface*. The term *router* refers to any machine running a routing protocol to make routing-table entry decisions (we explain routing protocols in Section 1.2). A *routing table* is a matrix, and for each packet passing through a router and directed towards a given destination, the router's routing-table entry determines which of the outgoing interfaces the packet should be forwarded along. A *packet* is a small piece of data to be transferred across a network. Each communication between two computers is broken into packets. Each router in an AS uses the same routing protocol. The term *host* technically refers to any computer connected to a network. However, throughout this study, we use the term host to refer to a special case of a router, with respect to routing, since both a host and a router maintain routing tables and make routing decisions based on packet destination; the only distinction is that a host does not receive transit traffic (i.e., it only generates or receives traffic, it does not forward traffic).

Each direction on a link has an associated *cost*, or *weight*, which is an estimate of the delay that a packet incurs while traversing that link. The method to determine this value is specific to the routing protocol being employed, e.g., the Routing Information Protocol (RIP) uses a cost of one for each link in the path [Hedr88]. Although opposite directions on a link often have the same cost, these costs may be different.

Each link is assumed to be bi-directional, and the routers at either end of a link are referred to as *immediate neighbors* [Come95].

When a packet is sent from one computer to another, it traverses zero or more routers. The sequence of routers that a packet traverses is termed its *route*, or *path*. The traversal of one router is called a *hop*. At any point in a packet's transmission, the next router to be traversed in the packet's route is termed the *next hop*. The *cost* of a route is the sum of the costs of the traversed links that make up the route. A route is considered *shortest*, or *optimal*, when it has the lowest cost of all the possible routes between the packet's source and destination. The data in a transmission from one host to another is termed a *data packet*, while the information in a transmission for the purpose of inter-router communication is termed a *control packet* [Come95]. A *flooded* transmission is one where the data is sent along every link in the network (reaching every node). First, the data is sent along all the links from the source, then each node receiving the transmission for the first time, sends the packet to each of its neighbors (excluding the neighbor it received the packet from). This process continues until the data reaches every reachable node in the network. Loops are avoided, since each router only re-transmits a packet when received for the first time, and then only along the links not yet traversed by the packet. Flooding is fast and is the most robust form of communication possible in a network containing faulty nodes [Perl88].

### 1.1.3 Objectives and Contributions of this Work

The objectives of this research are to identify and analyze the vulnerabilities of routing protocols, and to propose a network monitoring scheme to detect attacks that exploit these weaknesses. We present a protocol that will allow each AS to monitor, detect, and react to routers that drop or misroute packets. A router is considered to have *dropped* a packet when it is expected to forward that packet to the next hop and it does not. A router is considered to have *misrouted* a packet when it forwards the packet to a next hop that is not part of a shortest route with respect to that packet. Dropping or misrouting packets can be fault-related or can be the result of an attacker maliciously subverting a router and implementing a malicious routing protocol. The problem of detecting routers that drop or misroute packets (referred to as *bad routers* in this work) is interesting because it is independent of the problems that most secure routing protocol designs address [Cheu97, Finn88, HPT97, IoBl93, KuCr93, Malk93, MuBa96, Perl88], that is, the problem is not resolved by the addition of confidentiality, authentication, and/or integrity in routing protocol updates (a type of control packet, defined in Section 1.2) and data transmission.

Our detection scheme is based on the principle of conservation of flow in a network, a property which states that all packets sent into a node, and not destined for that node, are expected to exit that node [ChLe97]. We provide a protocol that tracks this flow, with respect to network traffic into and out of routers, and detects routers disrupting that flow. We refer to this protocol by the acronym WATCHERS: **W**atching for **A**nomalies in **T**ransit **C**onservation: a **H**euristic for **E**nsuring **R**outer **S**ecurity.

WATCHERS allows an AS to monitor its routers. It is distributed, with each router in the AS diagnosing only its immediate neighbors. That is, every router in the AS concurrently runs its own copy of WATCHERS, using logs from neighboring routers as input. A router executing WATCHERS is termed a *testing router* – with respect to its neighbors. A testing router diagnoses each of its immediate neighbor(s) which, with respect to this router, are called *tested router(s)*, using the WATCHERS *detection engine*, which has two parts: (1) log validation and (2) conservation-of-flow analysis. The WATCHERS detection engine outputs link identifiers for the links it determines to be shared with bad routers, and the testing router severs these links.

We make the following assumptions about the AS:

- *GOOD NEIGHBOR RULE:* Any bad router is neighboring at least one good router.
- *GOOD PATH RULE:* Each pair of properly functioning routers is connected by at least *one* path of properly functioning routers. This minimal amount of good router connectivity is necessary if the network is to remain connected when all bad routers have been removed.

WATCHERS is a significant improvement over other network monitoring protocols in four important ways. First, the implementation of WATCHERS can be fully known to the attacker; yet it can still maintain its effectiveness. That is, the attacker does not gain any ability to circumvent WATCHERS by knowing how it works. Many other network monitoring protocols (e.g., traceroute [ChBe94] and Simple Network Management Protocol (SNMP) [CFSD90]) can be circumvented in this manner; i.e., by knowing how the monitoring works the bad routers can act good to avoid detection. Second, WATCHERS can detect powerful malicious routers, including those routers which intelligently drop and misroute packets, as well as consorting (defined in Section 2.2.4) malicious routers. We know of no past work in network monitoring that can claim this degree of success. Third, WATCHERS can pinpoint the exact router(s) responsible for dropping or misrouting packets. While some network monitoring techniques can narrow the scope of suspects for malicious activity, most can only detect that a problem exists, and offer minimal help as to how to pinpoint the malicious router [CFSD90, GaSp96, ChBe94, Perl88]. Fourth, we formally prove that WATCHERS limits the number of false alarm diagnoses.

### 1.1.4 Organization

The remainder of this chapter introduces the basic concepts related to the disruptive router problem. Section 1.2 gives a brief overview of routing protocols. Section 1.3 introduces router vulnerabilities, discusses the properties that facilitate those vulnerabilities, and illustrates the need for our research. Finally, Section 1.4 outlines the structure for the remainder of the thesis.

## 1.2 Routing Protocols

Routing is a function of the network layer within the Open System Interconnection (OSI) model [Come95]. A routing protocol employs a distributed algorithm to keep routing tables up-to-date with the topology changes of the network. Every router within an AS runs the same routing protocol. In general, a routing protocol works in the following manner. Each router discovers and advertises – to the other routers in the AS – topological information (*update messages*) about the network. The router employs these update messages as input to a "best" (usually shortest) path algorithm to calculate route information, and then it stores that information in its routing table. The routing table is a matrix with each row containing the information needed to forward packets on to reachable destinations; each entry includes destination, route cost, and next hop (see Figure 1.1).

The "*best*" *path algorithm* (typically shortest path first, e.g., Open Shortest Path First (OSPF) [Moy94], and Routing Information Protocol (RIP) [Hedr88]) takes the current routing table and the update message(s) from other routers as input, and outputs a new routing table reflecting the changes in the topology introduced by the update messages. When a router has no remaining routing updates to process, its routing table is considered up-to-date, and the router once again uses this table to make forwarding decisions for the packets it receives. The two main methods used to propagate routing updates are the link-state and the distance-vector algorithms [Dick94, Huit95].

A link-state algorithm is based on the principle that, with a snapshot of the network topology, each router can independently decide on the best routes for its packets. This snapshot of the topology is maintained using adjacency updates: each router broadcasts updates advertising the operability status of its immediate neighbors (i.e., whether they are up or down). Thus, topological information is propagated in a distributed fashion but route computation is performed by each router independently [Come95, Dick94, Huit95, MRR80, Moy94].

A distance-vector algorithm is more simplistic. Each router sends and receives updates to and from immediate neighbors. These updates represent snapshots

**A's Routing Table**

| Destination | Cost | Next Hop |
|---|---|---|
| 10.0.0.0 | 0 | direct |
| 20.0.0.0 | 0 | direct |
| 30.0.0.0 | 1 | B |
| 40.0.0.0 | 1 | B |
| 50.0.0.0 | 1 | C |
| 60.0.0.0 | 2 | B |
| 70.0.0.0 | 2 | B |
| 80.0.0.0 | 2 | C |
| 90.0.0.0 | 3 | B |

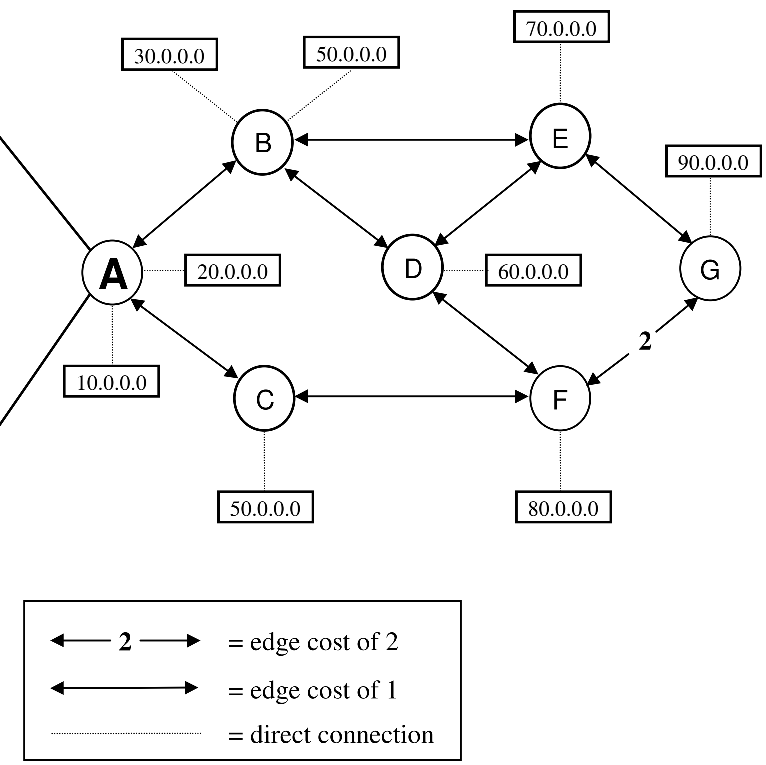



**Figure 1.1 Router A's routing table in a seven-router network.**

of the sender's routing table. The cost of the interface on which the update was received is added to the path costs in the sender's routing-table entries. If any computed path cost is lower than the corresponding path cost used in the current routing table, then it is updated [Come95, Hedr88]. Thus, route computation is performed in a distributed fashion.

Although WATCHERS can be used with either type of distributed routing algorithm, the link-state algorithm is generally considered more secure, since its computation is based on topological information and not just raw computations provided by other routers, as is the case with the distance-vector algorithm [Perl88].

## 1.3 Router Vulnerabilities

Routers and their associated protocols attract two classes of attack: those aimed at altering the topology of the underlying network (*topology attacks*) and those aimed *only* at adversely affecting host data in transmission (*data attacks*). Alternatively these two classes can be viewed as (1) an attack *by* a bad router *against* one or more good router(s) and (2) an attack *by* a bad router directly *against* a packet in the bad router's possession. The properties that make these attacks possible can be grouped into three categories.

- *A routing protocol's computation of routes* is dependent on a router's trust of the update messages it receives from other routers, but this approach is not secure since these messages typically do not provide confidentiality, integrity, or authentication (CIA) [Bell89, Pace97, PWMO97].

- *A routing protocol lacks CIA* in data packets, allowing transmissions to be viewed, altered, or copied in transit. The lack of authentication further complicates the problem since it reduces the possibility of accurately pinpointing disruptive router(s).

- *A routing protocol has no provision for monitoring* the routers within each AS. Although adding CIA capabilities to a routing protocol can be shown to solve many of its security problems [Cheu97, KuCr93, MuBa96, SMG97], only careful monitoring of the packet flow within an AS can detect faulty or malicious activity aimed at disrupting or terminating packet transmission.

Traditionally, research on router security has concentrated on the first two classes of vulnerability [Cheu97, Finn88, HPT97, IoBl93, KuCr93, Malk93, MuBa96, Perl88]. The objective of this research is to develop a design aimed at providing the third element of router protection, namely network monitoring. Two concepts motivate the need for this type of protection.

- *FAULTY ROUTING PROTOCOL – A routing protocol that does not work as its specification dictates.*

- *BAD ROUTER – A router running a faulty routing protocol.* A router that drops packets (*i.e., a black-hole router*), or misroutes packets must be employing a faulty, misconfigured, or malicious implementation of the routing protocol. A correct implementation of a properly working routing protocol will not generate such actions. A *compromised* router refers to a router that is trusted by the other routers in the AS (i.e., it can authenticate its transmissions), but has been subverted and, as a result, may be running a malicious implementation of the routing protocol. A *faulty* router is one in which the implementation is either misconfigured or incorrectly coded. An *unauthorized* router is a non-trusted router (i.e., it cannot authenticate its transmissions) that may be running any implementation of a routing protocol. All three types are considered *bad routers* with respect to this research, i.e., they can generate the disruptive activity that WATCHERS detects.

A topology attack takes advantage of the distributed nature of routing-table computations. For example, if a given router is compromised and sends an update message claiming that it has better routes than it actually does, these updates will be trusted and will affect the routing tables of routers throughout the network. Topology attacks create multiple problems, including packets being routed to compromised routers (denial-of-service), sub-optimum routes through the network, and isolated routers (i.e., routers which are not being used to transmit packets by any of their neighbors). These updates can also be sent within a masquerade, implicating the impersonated router [Finn88].

A data attack may occur whenever data traverses a compromised router. A given router can essentially do whatever it likes to a packet in its control. A data attack includes a packet being altered, a packet being viewed in transit, a packet being spuriously injected into the network, a packet being misrouted (denial of service), and a packet being dropped [Finn88]. All of these attacks are important to detect, but most can be handled by protocols that provide CIA [Cheu97, Finn88, HPT97, IoBl93, KuCr93, Malk93, MuBa96, Perl88]. Still, some attacks, e.g., misrouting and packet dropping, are possible even with CIA measures in place. This research focuses on enabling an AS to protect itself against these special types of attack.

## 1.4 Thesis Organization

Chapter 2 reviews the current methods used to provide network layer security, and introduces some attack scenarios to demonstrate the various attacks we are striving to detect. Chapter 3 presents WATCHERS, our distributed network monitoring protocol for detecting routers that drop or misroute packets. The key to pinpoint detection is for each router to monitor its local and immediate neighbor(s)' transit traffic. The results of this monitoring can be used to analyze conservation of flow with respect to network traffic. Any breakdown in flow can be used to pinpoint malicious routers that are causing packet dropping. Chapter 4 gives our concluding remarks and topics for further research.

# Chapter 2

# Background

Many mechanisms have been added to network architectures for isolating the source of faults in a network. In Section 2.1, we discuss the most marked examples of router monitoring techniques; then in Section 2.2, we describe the attacks WATCHERS strives to detect.

## 2.1 Current Rou ter Monitoring Techniques

### 2.1.1 Hop-by-Hop Acknowledgments

Perlman first proposed the idea of intermediate hop acknowledgments (ACKs) [Perl88]. Under this scheme, a packet being sent from router S to router D will not only get an ACK from D, but also get an ACK from all routers on the path from S to D.

Now, if a black hole is present on the path from S to D, we can theoretically isolate the bad router to one of two possibilities: either the router which did not send an ACK back to the source, or the router previous to that router in the path to the destination. We illustrate these two cases, respectively, in Figure 2.1 and Figure 2.2. However, in the case where malicious *or* faulty routers are in the network, good intermediate routers may be easily implicated by bad intermediate routers, resulting in false positive diagnoses, as illustrated in Figure 2.3 [Perl88].

### 2.1.2 Protection from Packet Corruption

A router can alter the header of a packet in such a way that it may be dropped or misrouted. If, for example, a packet's destination field is changed in transit, that packet will be misrouted (with respect to its intended destination). This type of attack illustrates the need for protecting packet integrity using checksums and authentication. Unfortunately, as we have already shown, packets may be dropped in transit using methods other than packet alteration. Therefore, CIA-based protection is necessary but not sufficient. For examples of this type of router infrastructure security, we refer the reader to [Cheu97, Finn88, HPT97, IoBl93, KuCr93, Malk93, MuBa96, Perl88].

A related method using hop-by-hop checksum verification was proposed in [Perl88]. The checksum is verified at each hop, in an attempt to isolate a faulty packet-corrupting router. This method is both time intensive and open to malicious attack. The simplest example to defeat this method is to corrupt the packet and regenerate the checksum. Hence, it is not a realistic solution for a malicious routing model.

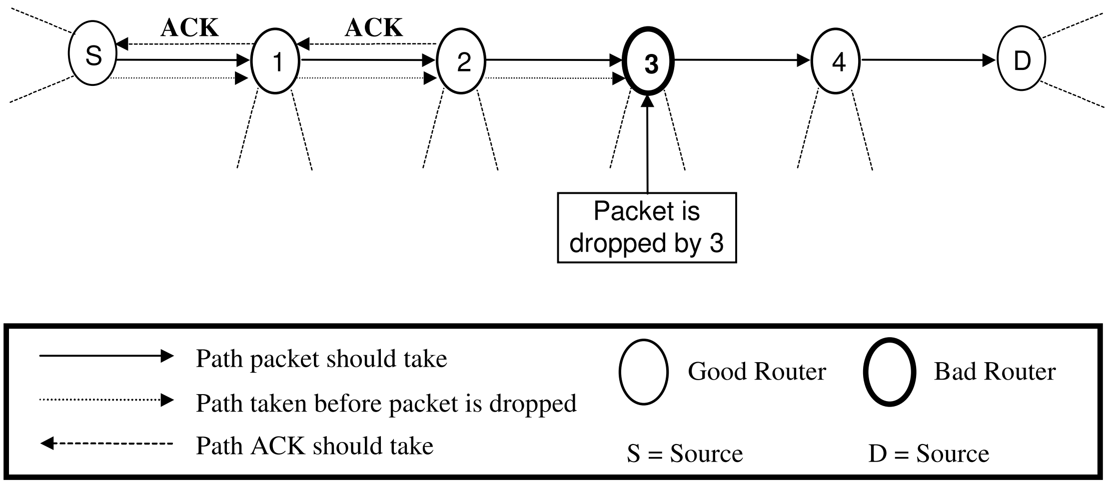



**Figure 2.1 Router 3 drops the packet, and does *not* send an ACK.**

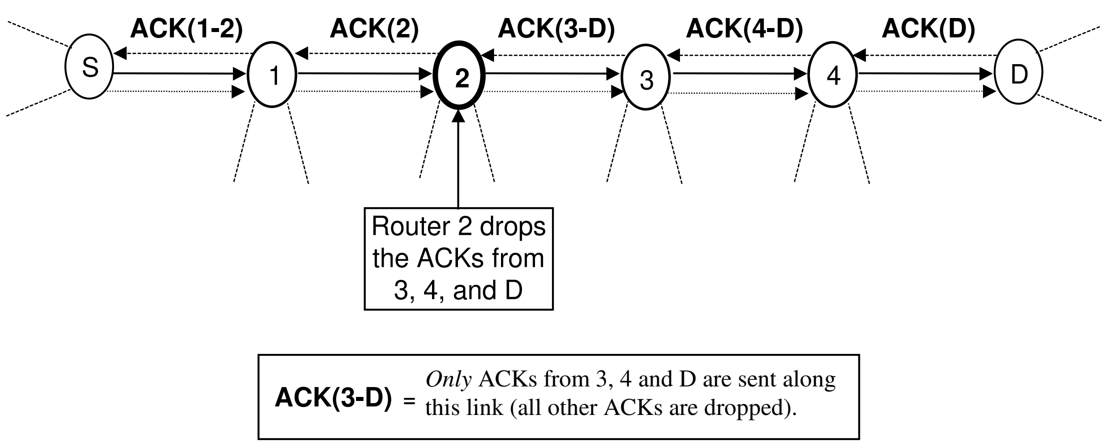



**Figure 2.2 Router 2 drops all ACKs, but sends the packet.**

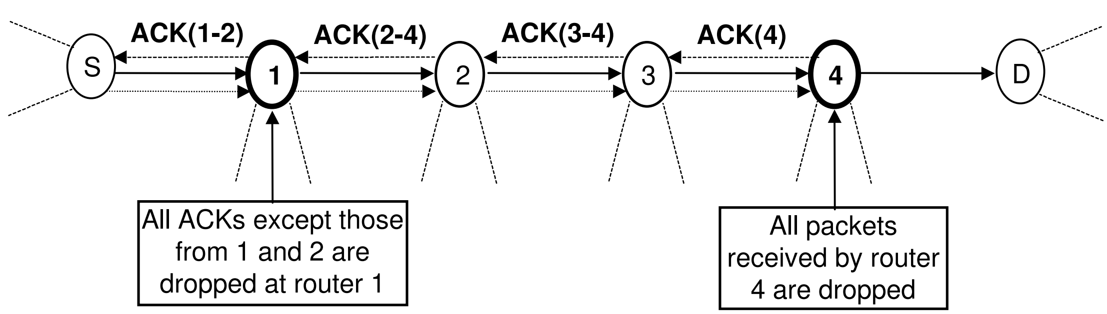



**Figure 2.3 Routers 1 and 4 consort to wrongly implicate routers 2 and 3.**

### 2.1.3 Probing Techniques

Probing techniques are based on peer testing of routers. Cheung proposes a router-monitoring protocol based on a probing technique [ChLe97]. The idea is to test each hop of a suspicious route, starting with the closest hop. A probe is sent to the first hop on the route. A *probe* is a packet whose content is such that it will use a pre-determined path that includes the testing router itself (this could also be called a *round-trip packet*). If the probe packet returns, the next probe is sent *through* the first (already-tested) hop to the second hop. If this second probe packet returns, then the next probe is sent *through* the first and second hops to the third hop, and so on until a probe is not returned or the destination is reached. Figure 2.4 illustrates this scenario. If a probe does not return, then the furthest intermediate router from the testing router is considered faulty or malicious.

Although Cheung's probing method has applications in the area of purely faulty networks, probing has two principal limitations in a malicious environment. First, a testing probe is easy for a malicious router to detect and treat as a special case. Second, a probe does not always accurately emulate real traffic. In order to claim that a router *is good* with respect to the traffic it handles, the traffic used to test that router should accurately emulate real traffic. Since probing cannot guarantee representative traffic, it may allow bad routers to be diagnosed as good (*false negative diagnoses*).

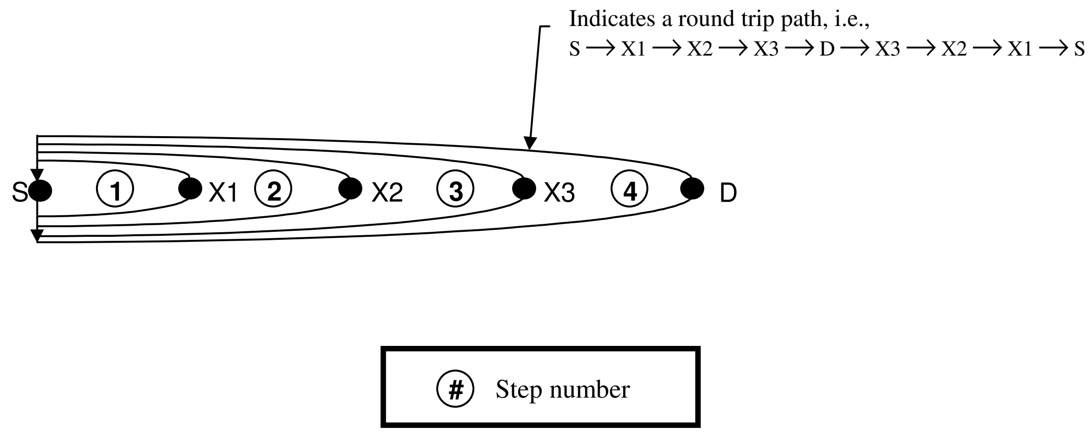



**Figure 2.4 Probing method using round trip testing of each router in a path from S to D.**

### 2.1.4 Network Management

A network management tool is a distributed service that collects data from all the routers in an AS, and communicates with other network management modules in order to exchange and compile information used to debug problems, control routing, and find computers that violate protocol standards [CFSD90, Come95]. Unfortunately, current examples of the network management model, typified by SNMP, do not provide access to all of the events necessary to monitor transit traffic within a network [Blac94, CFSD90, Held96]. We propose that transit traffic information is necessary to detect packet-terminating routers.

### 2.1.5 Re cording and Tracing Routes

Route recording and route tracing are often the first methods used to isolate faults in routers [ChBe94].

Route recording is typically optional, e.g., in the Internet Protocol (IP). When a packet has this option set, each router on the path to the packet's destination appends its IP address to a list in the packet's header. Route recording is a poor way to isolate malicious routers due to three important reasons. First, the source *and* the destination must agree in advance to send and receive recorded routes; if the destination is not informed, it will disregard the route list. Second, the list can be altered in transit by one or more of the intermediate routers, making any results extrapolated from this option unreliable. Third, the packet can be dropped, leaving little recourse for the router that initiated the recording [Come95].

Route tracing (e.g., the traceroute program [ChBe94]) is also an optional mechanism, typically used on a per-packet basis to manually test the routing infrastructure. This approach is similar to hop-by-hop acknowledgments except that each hop is tested using a separate packet. Thus, the mechanism has the same problems as hop-by-hop acknowledgments with the added strain on network load [Come95, Perl88].

## 2.2 Attack Scenarios

### 2.2.1 Overview

Many circumstances can result in a router disrupting traffic. These circumstances range from natural faults to misconfiguration to malicious intent. This section provides a comprehensive examination of the attacks WATCHERS is designed to detect.

We start by discussing simple packet-dropping routers, and from here we expand to consorting bad routers and how each scenario can affect what we need for detection.

### 2.2.2 Black-Hole Router

A black-hole router drops packets instead of forwarding them on to their intended destination, as illustrated in Figure 2.5. In its most basic form, this attack does not discriminate, dropping all packets. Of course, this scenario simply mimics the characteristics of an inoperable router, and can therefore be detected using methods already in place (such as traceroute). However, this attack becomes difficult to detect if the attacker is intelligent (an *intelligent black hole*), correctly forwarding a viable percentage of the traffic, being careful not to delete control packets. In fact, this attack becomes nearly transparent, since neither the source nor the destination is notified of the location where the packet was lost. Further, under TCP/IP, when the source's ACK wait timer expires, waiting for a packet to be acknowledged, the source will only be able to speculate as to what route the packet was using when it was lost. With the responsibility of recourse left to the source, there is little chance that the black hole will be pinpointed.

The term black hole has been extended to also include routers that lure packets by issuing disingenuous topology updates. These updates falsely advertise optimal routes, causing the surrounding routers to forward packets on to the compromised router.

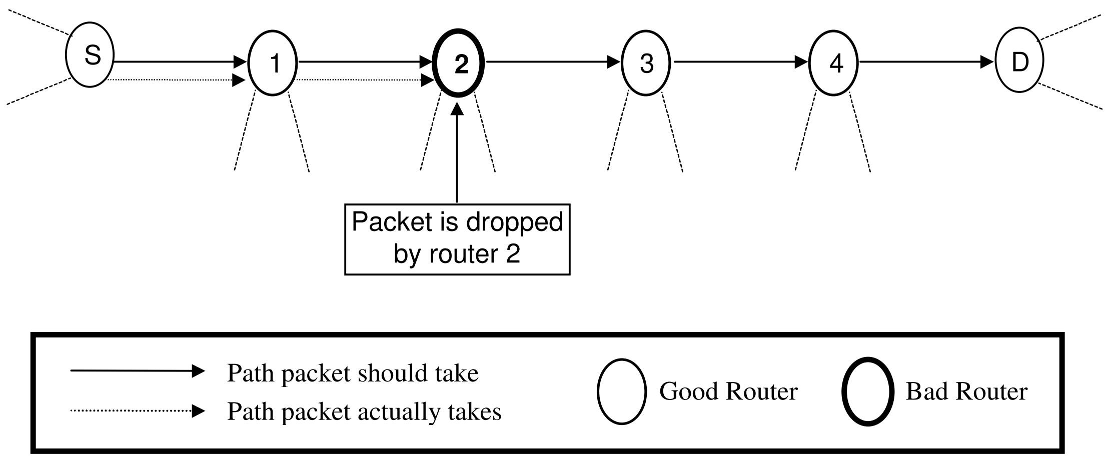



**Figure 2.5 Simple lone black hole.**

### 2.2.3 Examples of Intelligent Black Holes

To this point, we have been referring to black holes as routers that drop *all* of the packets they receive, but black holes can be *intelligent*, i.e., they can also selectively drop packets. Some examples are:

- *EPISODIC* – drop packets occasionally, perhaps a percentage of the time.
- *CONTENT AWARE* – drop packets based on their content, perhaps using the packet's source and/or destination address.
- *DETECTION AWARE* – drop packets using methods meant to elude detection, perhaps employing consorting router alliances such as those explained below.

### 2.2.4 Consorting Routers

When designing a defense against black-hole routers, it is critical to take into account the possibility of consorting bad routers. A *consorting bad router* is a router that conspires with another bad router to hide the faulty behavior of that router; covering for, or aiding in, its malicious activity. Consorting bad routers can be neighboring or separated by one or more good routers. They are more powerful than lone bad routers. This should be clear, since consorting bad routers can use advanced methods to cover one another's tracks. In fact, we have already seen the effectiveness of malicious routers consorting in the attack on the hop-by-hop acknowledgment method illustrated in Figure 2.3. In that attack, router 4 dropped a packet while router 1 covered for router 4 by dropping ACKs from those routers beyond router 2. This situation wrongly implicates (i.e., a *false positive diagnosis*) routers 2 and 3, i.e., either router 2 dropped the packet, but sent an ACK; or router 3 dropped the packet, and sent no ACK. Without a consorting partner, router 4 would not have been able to avoid detection.

In Chapter 3, we present methods by which bad routers can consort to fool a simple conservation-of-flow protocol, and we discuss how the WATCHERS protocol counteracts these attacks.

# Chapter 3

# The WATCHERS Protocol

## 3.1 Overview

In this chapter, we describe the WATCHERS security protocol and provide concrete examples showing how it detects disruptive routers. WATCHERS has four key properties that make it a powerful detection protocol. First, WATCHERS is distributed, with each router diagnosing its own neighbors, helping it scale to large networks. Second, WATCHERS is robust in the face of misinformation from bad routers, because it uses redundant logging that allows each router to perform validity checks on the in-

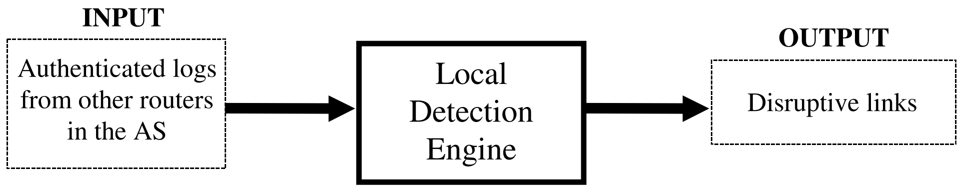



**Figure 3.1 High-level model of WATCHERS running at each router in the AS.**

formation received from other routers. Third, WATCHERS is resistant to false positive and false negative diagnoses, since information from throughout the network is used to monitor malicious activity. Finally, WATCHERS can pinpoint bad routers in networks with consorting bad routers, whether they are neighboring or not.

## 3.2 High-Level Description

At the highest level, WATCHERS is a distributed network monitoring tool. Each testing router logs counters that represent incoming transit traffic and outgoing transit traffic with respect to the links between itself and its immediate neighbor(s). We will leave the specifics of maintaining these counters for Section 3.3. What is important here is that each testing router keeps these counters for both itself and redundant counters, used for verification, for each immediate neighbor. This allows the algorithm to compare the *perspectives* of immediately neighboring routers. With these counters, it is now possible to monitor the traffic flow into and out of each node in the AS. The algorithm consists of three levels of detection. First, each testing router initiates the protocol and makes a call for logs, detecting those routers that refuse to participate. Second, the testing router checks the validity of the counters. Finally, for each tested router that passes the previous tests, the testing router analyzes its conservation of flow to verify that the expected flow through the router matches the actual flow through the router. If any one of these tests fails, the testing router severs its neighboring relationship with the router being tested.

As Figure 3.1 illustrates, each WATCHERS testing router uses authenticated log messages from routers in its local topology (those within two hops; as explained in Section 3.4). These logs are used as input to the router's WATCHERS detection engine, which returns link identifiers as output. These link identifiers represent the disruptive links (those directly connected to disruptive routers) that will be severed by the testing router.

WATCHERS assumes that we have no control over the actions of bad routers. We will show that, even in the presence of consorting bad routers, WATCHERS eventually removes all links connecting good and bad routers, therefore logically removing all bad routers from the AS topology.

In the remainder of this chapter, we expand on the mechanisms of the protocol, including how they work, why they work, and why each was employed. We describe the protocol in the order specified in Figure 3.1: first input, then detection, and finally output.

## 3.3 Input – Logging at the Router

We log two types of information: transit traffic and misrouted traffic.

### 3.3.1 Logging Transit Traffic

In an AS running WATCHERS, each router is responsible for locally logging the transit traffic that goes through itself. This logging is done using a counter for each type of transit traffic: transit for local, transit for neighbor, and transit for both.

Figure 3.2 illustrates the three types of transit traffic, presenting each type from router X's point of view. The first, $D_{X,Y}$, refers to the amount of traffic transit *through* X but destined for router Y. The second, $S_{X,Y}$, refers to the amount of traffic sourced at X but transit *through* Y. The third, $T_{X,Y}$, refers to the amount of traffic that is transit *through* both X and Y.

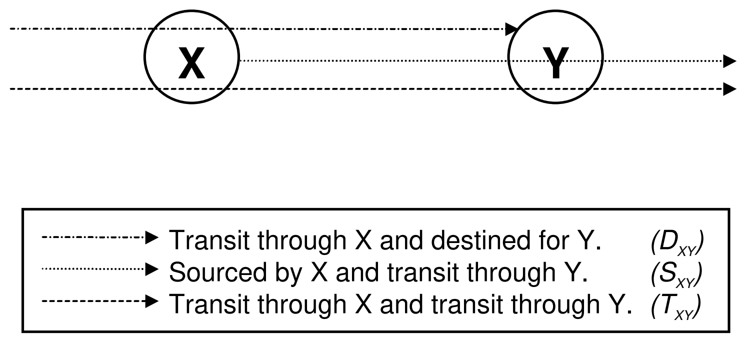



**Figure 3.2 Transit traffic for interface $X \rightarrow Y$.**

Each of these transit traffic types can be viewed from two "perspectives": X's and Y's. We use the concept of perspectives because X and Y may have differing opinions as to what categories their shared transit traffic fall into.

So, to summarize, for each interface we have three counters for each direction of traffic flow. For the $X \rightarrow Y$ traffic flow, we have $D_{XY}$, $S_{XY}$, and $T_{XY}$; for the $Y \rightarrow X$ flow, we have $D_{YX}$, $S_{YX}$, and $T_{YX}$. Further, note that routers X and Y *each* maintain local copies of these six counters, creating a natural redundancy, which will be exploited later in validating the logs. We illustrate this redundant method of storage in Figure 3.3.

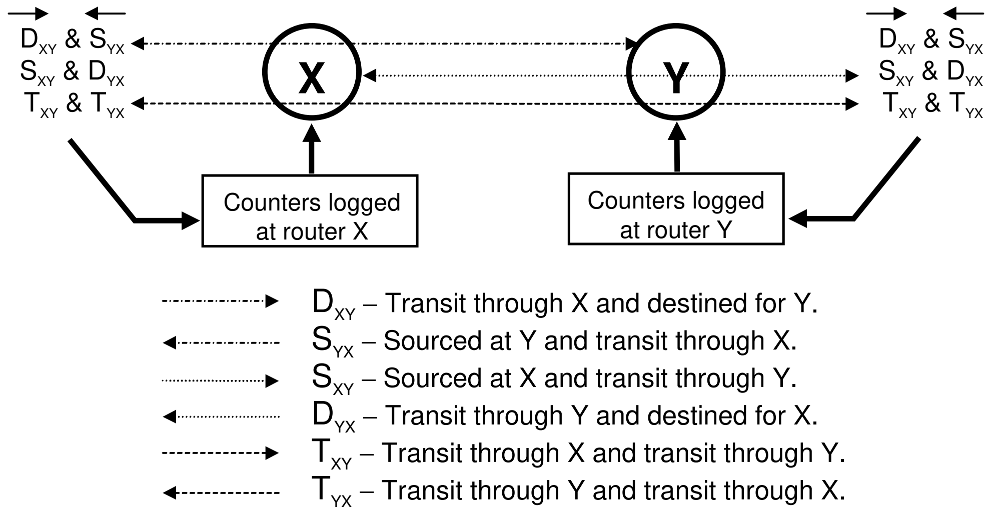



**Figure 3.3 Counters logged at each endpoint of the interface $X \leftrightarrow Y$. Note that each counter for this interface is redundantly logged at both endpoints.**

### 3.3.2 Logging Misrouted Packets

In addition to logging transit traffic, a router running WATCHERS must keep a log of the number of times each of its neighbors misroutes traffic. For each neighbor, we need another counter, $M_{neighbor,local}$, e.g., router Y would increment $M_{XY}$ to keep track of the number of times router X wrongly, with respect to the optimal path, forwards a packet to Y. That is, Y increments $M_{XY}$ every time router X misroutes a packet to router Y. Logging instances of a neighbor, say router X, misrouting packets requires that the logging router, Y, maintain a local copy of X's routing table (see Figure 3.4). Using this local copy of X's routing table, Y verifies each packet forwarded by X. Any discrepancy results in Y incrementing $M_{XY}$.

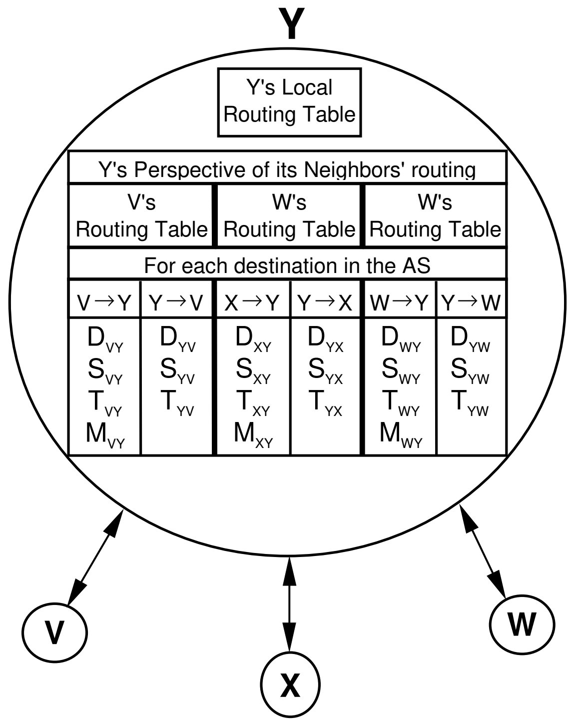



**Figure 3.4 Comprehensive view of the log maintained at router Y, with neighbors V, X, and W.**

Determining which routers are misrouting packets works on the assumption that *neighboring* good routers within the same AS agree on the network topology. This assumption is not presumptuous since good routers should send routing updates whenever there is a change to their routing tables. Furthermore, when one router, say X, sends an update, its neighbor, say Y, will receive that update in a nominal amount of time (since the two routers are directly connected). Thus, if Y is working properly, it will change its local routing table to agree with that of the sender, X. The time between the two routing-table changes should therefore be nominal [Huit95]. Moreover, in modern link-state routing protocols, link costs are static (not dynamically dependent on link load) making link-state changes infrequent [Stee95]. Even with nominal delay, neighbors may disagree occasionally, and therefore log traffic differently. For this reason, we suggest the use of thresholds that represent the level of faulty activity deemed acceptable, after which faulty activity is considered bad. We discuss thresholds in Section 3.6.

Misrouting detection works in the following manner. A router, say Y, receives a packet from a neighbor, say X, along the $X \rightarrow Y$ link, say link L. The packet is destined for node B, as shown in Figure 3.5. First, Y consults its locally-generated *perspective* of X's routing table. If the next-hop column for a packet with destination B is *not* Y (the local router itself), then Y increments its $M_{XY}$ counter, indicating that X misrouted the packet, i.e., the packet should not have come to Y. When the logs are analyzed by the WATCHERS detection engine, high (above some acceptable[1] threshold) M-counter values will be treated as an indication that the sending router is misrouting packets.

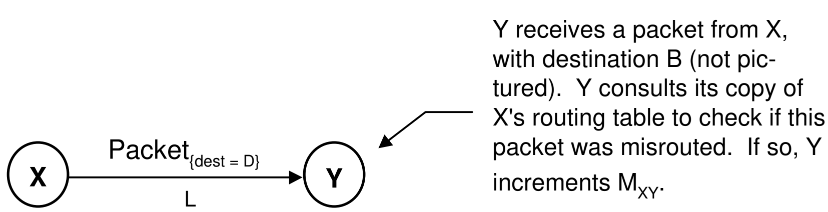



**Figure 3.5 Y checks if X misrouted a packet destined for B.**

### 3.3.3 Destination-Specific Logs

Given how the transit counters keep track of packet flow, it should be clear that they do not provide quite enough information to detect consorting routers (see

[1] Neighbors should be allowed to disagree occasionally since updates across the direct link may experience some delay. However, no two good neighbors should disagree very often or for very long.

Section 2.2.4). The counters, as currently implemented, do not provide any mechanism to track where incoming and outgoing packets are destined. We illustrate the problem with an example where a packet needs to be routed from source *A* to destination *B* (see Figure 3.6). Suppose the optimal path for the packet is $A \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow B$, but routers 3 and 4 are consorting to disrupt traffic. Now, say *A*, acting properly, sends the packet to router 1 (and both increment their $S_{A,1}$ counters). Then, router 1 sends the packet to router 3 as expected (both increment their $T_{13}$ counters). But, now router 3 sends the packet to consorting neighbor 4 and both *falsely* log the packet by incrementing their $D_{34}$ counters. The attack is completed when router 4 drops the packet(making it appear that the packet was destined for router 4). These operations appear legal to the neighbors of routers 3 and 4, because routers 3 and 4 have *consorted* in their claim that the packet dropped by router 4 was destined for router 4 instead of the actual destination, router *B*. In short, routers 3 and 4 are able to consort to disrupt traffic without being detected by their neighbors, since they are able to log the traffic to appear like it was destined for router 4.

The problem is that the current counters carry no information about the traffic's destination, thus there is no way to refute false logging by neighboring consorting routers.

To detect consorting bad routers, we add destination-specific counters to the logs. Each router maintains a set of transit counters *for each destination within the AS*, to permit monitoring flows even through bad routers. To illustrate the benefits, we will step through the previous example again, this time using destination-specific counters (see Figure 3.6). The first hop of a transmission from router *A* to router *B* is from router $A$ to router 1; it increments $S_{A,1}[B]$ (which can be read as "a packet generated at router $A$ and transit through router 1 with destination $B$") at both router $A$ and router 1. Notice that the only difference in the notation is the $[B]$ portion, indicating that the counter is only incremented for packets with destination $B$, i.e., the counter is destination-specific. The second hop of the transmission, from router 1 to router 3, increments $T_{1,3}[B]$. Now, if router 3 sends the packet to consorting neighbor 4, these two bad routers, 3 and 4, cannot avoid detection by both incrementing $D_{3,4}[B]$, because there is a contradiction since the destination cannot be both router 4 *and B*. If router 3 did make this claim, it would be detected by all its good neighbors – i.e., routers 1, 2, and 5 – and be marked as a bad router.

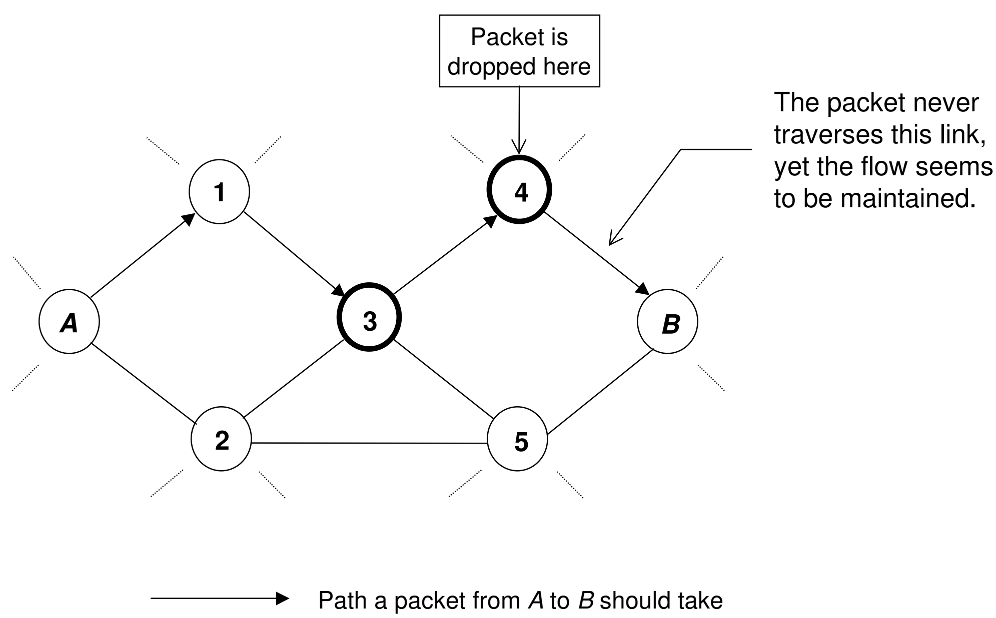



**Figure 3.6 Routers 3 and 4 consorting to drop a packet and foil detection.**

### 3.3.4 Pseudo-Code for Incrementing Counters

Figure 3.7 provides the pseudo-code showing how each of the counters should be incremented. Deciding which counters to increment requires only two extra compare operations per packet, resulting in minimal extra load at the router.

Although it is easier to think of incrementing the counters in single unit blocks, these counters are instead incremented by the size of the data contained in the packet. This is to accommodate conventional network practices, such as fragmentation[2], where one packet is broken up into multiple smaller packets, or reassembly, where multiple packets are assembled into one larger packet.

## 3.4 Detection Engine – Gathering Input and Performing Diagnoses

The detection engine portion of WATCHERS has two sub-protocols. First, the request, respond, and receive sub-protocol is used to initiate WATCHERS and to detect those routers which do not participate with the protocol. Second, the validation and conservation-of-flow algorithms are used to diagnose neighboring routers as bad or good. This diagnosis part of the WATCHERS protocol is described in the remainder of this section. The entire WATCHERS protocol is summarized as pseudo-code at the end Section 3.4.

[2] For example, if a packet containing 10,000 bytes is sent on path $A \to B \to C$, it might be transmitted as ten 1000 byte packets along link $A \to B$, and then be *fragmented* at router B and sent as twenty 500 byte packets along link $B \to C$.

```
if INCOMING PACKET (Y → X) then
        /* Check if Y misrouted this packet */
        if Y->NextHop[d] != X then
                M_YX[d]++                          misrouted by Y
        endif

        /* Increment transit counters for an incoming packet */
        if Y is the source then
                if X = d then
                        do nothing                 not transit traffic
                else
                        S_YX[d]++                  transit through X
                endif
        else if X = d then
                D_YX[d]++                          transit through Y
            else
                T_YX[d]++                          transit through Y&X
            endif
        endif
endif

if OUTGOING PACKET (X → Y) then
        /* Increment transit counters for an outgoing packet */
        if X is the source then
            if Y = d then
                do nothing                         not transit traffic
            else
                S_XY[d]++                          transit through Y
            endif
        else if Y = d then
                D_XY[d]++                          transit through X
            else
                T_XY[d]++                          transit through X&Y
            endif
        endif
endif
```

Note that router X needs to check if it is the destination anyway, so this is not a new operation

*This test can be avoided:*

In the case when X generates the packet, it can go straight to this test.

In the case when X receives the packet, it can go straight to this test.

**Figure 3.7 Pseudo-code showing how to increment counters at router X when *d* is the packet's destination.**

### 3.4.1 Request, Receive, and Respond

Diagnosis with WATCHERS is meant to be executed periodically. We call each periodic execution of WATCHERS a *phase*. Each phase is meant to be AS-wide, with all routers running the protocol concurrently [ChLe97]. The request, receive, and respond (RRR) log dissemination sub-protocol makes synchronized concurrent AS-wide phase initiation and log retrieval possible. RRR requires two additional restrictions on the AS model. First, good routers must be in the majority. Second, good routers should have equivalent clock-tick rates.

RRR initializes WATCHERS in three stages: local initiation, AS-wide router synchronization, and AS-wide initiation. In the first stage – *local initiation* – each router periodically initiates WATCHERS independently of the other routers in the AS. In the second stage – *AS-wide router synchronization* – each router which has already performed local initiation, floods synchronization messages (termed *request messages*) to every other router in the AS to communicate a desire to start a new phase of WATCHERS. The final stage – *AS-wide initiation* – occurs when each router has received a *request message* from a majority (over 50%) of the routers in the AS. After all three stages of RRR, initiation is complete and all good routers have entered a new phase of WATCHERS. Now, each router in the AS floods a message (called a *respond message*) containing its counters. When a router has received all of the counter values it requires for detection, it begins the *diagnosis stage* of WATCHERS, explained in Section 3.4.2. We expand on the three stages of RRR initiation below.

- *Local Initiation* – Each router keeps a *phase-start* counter marking the time elapsed since the last run of WATCHERS, and when the counter reaches a predetermined value,[3] a new execution of WATCHERS is locally initiated at the router.

- *AS-wide Router Synchronization* – Once a router has locally initiated WATCHERS, it floods a *request message* (see Figure 3.8), then sets a timer (the maximum time a router should wait for AS-wide initiation), and waits for *request messages* from all other routers in the AS. As the name implies, this stage is used to synchronize the routers, and takes advantage of the flooding algorithm's fast propagation rate.

- *AS-wide Initiation* – There are two possibilities for a waiting router:
  - *Successful initiation*: It receives authenticated *request messages* from a majority (over 50%) of the routers in the AS, at which time a new phase of WATCHERS is considered to have begun at every good router in the AS. This is where we use the good-router-majority assumption, i.e., if a majority of routers respond, then at least one is a good router, making the initiation valid.
  - *Failed initiation*: Its timer can run out, in which case it considers the AS-wide router synchronization to have failed. Failure occurs when less than a majority of routers respond to AS-wide router synchronization, thus

[3] All the good routers in the AS should reach this value at relatively the same time (assuming they experience equivalent propagation delays), since their clock-tick rates are equivalent and all their timers were initiated at the last AS-wide initiation. However, AS-wide router synchronization is designed to further synchronize the routers for AS-wide initiation.

protecting against the possibility of a false initiation of WATCHERS by bad routers.

At this point, every good router is expected to be executing the same phase of WATCHERS, and is required to flood a *respond message*, containing a digitally signed message digest (used for message authentication)[4], the phase number (a unique number indicating the phase in which the message was sent), and a copy of the local router's counters (as they were at the time the phase begun). The request and respond message contents are illustrated in Figure 3.8. After the counters are stored and sent, they are re-initialized to provide a fresh beginning for the next phase. Thus, each phase begins with fresh counters to reduce the effect of clock skew as well as counter wraparound. We discuss time-related counter skew in Section 3.6 in our discussion of the threshold mechanism.

When a router receives an authenticated respond message for the first time, it checks if the sender is one, two, or more hops away from it in the AS topology. If the first or the second case hold[5], then the router stores the message to be used by the diagnosis engine. If the third case holds, the receiving router does not store the message. In any case, the router retransmits the message in accordance with the flooding protocol, sending the message along every interface other than the one on which it was received.

---

[4] For information on digital signatures, see [Denn83, GaSp96].

[5] The sender is a neighbor or two-hop neighbor with respect to the receiver.

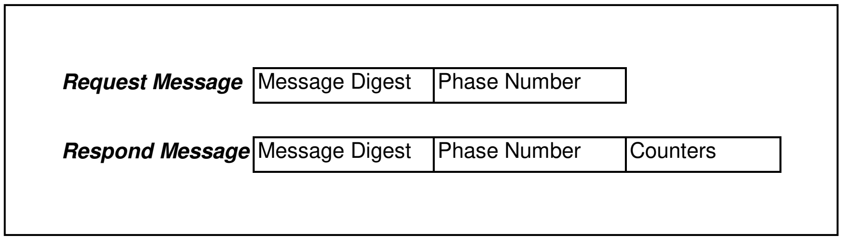



**Figure 3.8 RRR sub-protocol message contents.**

The final portion of the RRR sub-protocol has to do with authentication and the *good-path* assumption introduced in Section 1.1. Each request and respond message's integrity and authenticity will be verified using the digitally signed message digest. It should be clear that checking for authentication does not preclude denial-of-service attacks. Messages can be altered or dropped in transit, effectively denying the receiver the intended information. However, it should also be clear that under the *good-path* assumption, any respond message not received or not authenticated by a requesting router implicates the expected responding router, marking it as bad.

Once the logs are gathered and authenticated, we can assume that they are genuinely produced by the signing router, and they are given as input to the diagnosis algorithm. This allows the remainder of the algorithm to treat these logs as authentic. Any router marked as bad during the initiation stage is not considered during the diagnosis stage described in Section 3.4.2.

### 3.4.2 Diagnosis

The diagnosis algorithm has two parts: validation (*link diagnosis*) and conservation-of-flow analysis (*neighbor diagnosis*). Before explaining the algorithm, it is important to note that its correctness depends on two key properties.

- *Each testing router considers only itself to be good* – A router executing WATCHERS is *not* required to be good (i.e., WATCHERS does not give routers any new malicious powers), it need only trust itself. Any time a good *testing router* disagrees with a neighbor it can assume that the neighbor is bad since it trusts itself to be good. A bad router has nothing to gain by falsely diagnosing a good neighbor as bad, since such a diagnosis will result in severing their neighboring relationship (severing links between good and bad routers is the goal of WATCHERS).

- *Good testing routers are not hindered by bad routers refusing to run WATCHERS* – A bad router cannot be expected to run WATCHERS. A bad router not running WATCHERS does not hinder a good router from effectively doing so. At the start of every phase, a router expects to receive a respond message from each of its immediate neighbors (as well as its two-hop neighbors). If a router refuses to participate, it will be detected (and have its link severed) when its respond message is not received.

- *A bad router cannot misuse WATCHERS to implicate or affect the implication of another router as bad* – Refer to Section 3.9, lemma 3.9.1.

#### 3.4.2.1 Terminology and Concepts

A testing router diagnoses each of its immediate neighbor(s), the *tested router(s)*. The term *shared link* refers to the interface between a testing router and a tested router. The testing router performs diagnosis using its own logs, those of the tested router, and those of each of the tested router's neighbors. Each of the tested router's neighbors – excluding the testing router – are termed *two-hop routers* (since they are two hops away from the testing router), and a link connecting a tested router and a two-hop router is termed a *two-hop-shared link*. We illustrate these relationships in Figure 3.9.

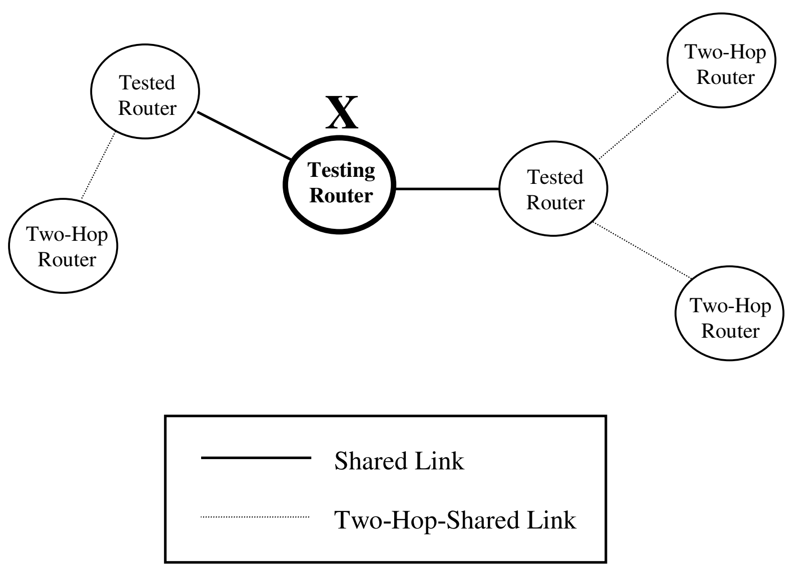
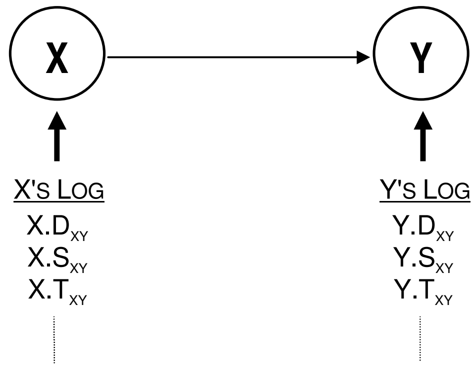



**Figure 3.9 Diagram of our terminology with respect to X as the testing router.**

#### 3.4.2.2 Validation

Each neighbor that passes initiation without being marked as bad, goes through validation analysis. The validation portion of the algorithm makes straightforward use of the redundant logging maintained by each set of neighbors. Intuitively, the idea is that two good neighbors should agree on the traffic that went across each direction of their shared interface. Testing for this agreement between the *testing router* and the *tested router* reduces to a simple comparison of their counter logs with respect to their *shared link*. As Figure 3.10 shows, a discrepancy between any pair of corresponding counters, larger than a pre-determined threshold (which can be 0, creating a zero-tolerance model), represents excessive packet dropping and results in the good router marking the *shared link* as bad.

There are two levels of validation: *local and two-hop*. *Local validation* checks *shared links*. *Two-hop validation* checks *two-hop-shared links*. If a *shared link* is found to be bad, it is logically severed by the testing router. On the other hand, a *two-hop-shared link* found invalid cannot be logically removed by the testing router, instead that decision is left up to the routers at either end of the link; this approach works well, since any good router connected to that link will make the same decision locally (i.e., it will be using the same counter values), and sever the link as the algorithm dictates.




| Model | Comparison | True | False |
|---|---|---|---|
| **Zero Tolerance** | $X.\{S,D,T\}_{XY} \neq Y.\{S,D,T\}_{XY}$ | X cuts off Y | OK |
| **Threshold** | $\lvert X.\{S,D,T\}_{XY} - Y.\{S,D,T\}_{XY} \rvert$ > THRESHOLD | X cuts off Y | OK |

Notation : $(X.\{\ldots\}_{XY})$ : Element of X's log.

**Figure 3.10 Validation test for the $X \rightarrow Y$ interface.**

Validity checking is not enough to pinpoint all bad routers. Bad routers can log counters in two ways after dropping a packet: they can keep accurate logs (*truth tellers*) or they can keep inaccurate logs (*liars*). The validation test only catches *liars*; to catch *truth tellers* we use conservation-of-flow analysis.

#### 3.4.2.3 Conservation of Flow

If the diagnosis passes through initiation and validation analysis, conservation-of-flow analysis is performed on the *testing router's* immediate neighbors (*tested routers*). However, unlike the validation test, here we need more than just the *tested router's* counters, we also need the counters from the *two-hop routers* directly adjacent to each *tested router*. For example, if router X in Figure 3.11 is the *testing* *router*, it performs a test on routers $Y_1$ and $Y_2$, its neighbors. To test the flow through $Y_1$, X would need counters from $Y_1$ and each of $Y_1$'s neighbors: $Z_1$, $Z_2$, and X itself. To test the flow through $Y_2$, X would need counters from $Y_2$, $Z_1$, $Z_3$, and X itself.

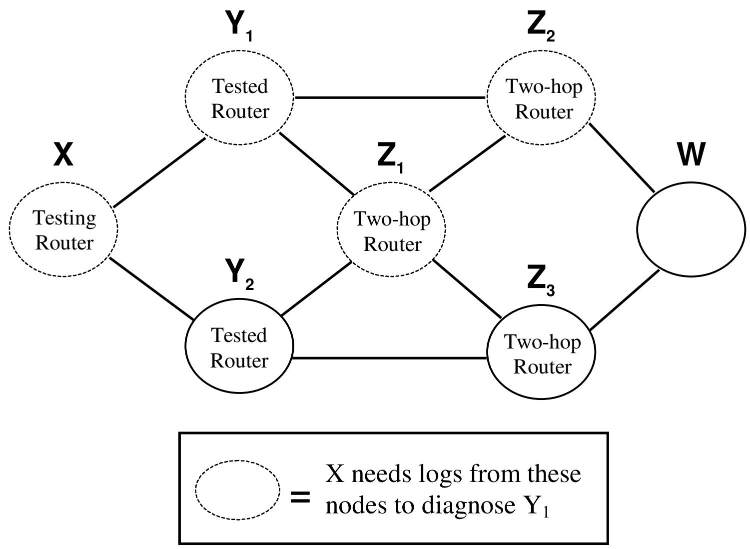



**Figure 3.11 Nodes involved in X's diagnosis of $Y_1$.**

Conservation-of-flow analysis is based on a simple principle: the amount of traffic that is *expected* to go through a router should match the amount of traffic that *actually* comes through the router. Suppose X, in Figure 3.11, wants to test $Y_1$.

The *expected* flow (or *incoming transit flow*) is determined using the counters for $Y_1$'s incoming edges, i.e., those tracking flow from routers X, $Z_1$, and $Z_2$. This incoming transit flow can be expressed in general by the following summation:

$$\sum_{\forall N \mid Y_1 \leftrightarrow N} \left( \mathrm{S}_{N,\mathrm{Y}_1} + \mathrm{T}_{N,\mathrm{Y}_1} \right)$$

The summation does not include $\mathrm{D}_{N,\mathrm{Y}_1}$ since packets incrementing this counter are destined for $Y_1$ and, therefore, not expected to flow out of $Y_1$.

The *actual* flow (or *outgoing transit flow*) is determined using the counters for $Y_1$'s outgoing edges, i.e., those tracking flow to routers X, $Z_1$, and $Z_2$. This outgoing transit flow can be expressed in general by the following summation:

$$\sum_{\forall N \mid Y_1 \leftrightarrow N} \left( D_{Y_1,N} + T_{Y_1,N} \right)$$

The summation does not include $S_{Y_1,N}$ since packets incrementing this counter are not transit through $Y_1$.

The difference between incoming transit flow and outgoing transit flow is illustrated in Figure 3.12, with expected flow shown as traffic going into $Y_1$, and actual flow being shown as flow coming out of $Y_1$. Now testing the conservation of flow is easy, since we just compare a neighbor's incoming transit flow to its outgoing transit flow:

$$\textbf{if} \left| \sum_{\forall N|N \leftrightarrow Y} \left( \mathbf{X.S}_{N\mathbf{Y}} + \mathbf{X.T}_{N\mathbf{Y}} \right) - \sum_{\forall N|N \leftrightarrow Y} \left( \mathbf{X.D}_{\mathbf{Y}N} + \mathbf{X.T}_{\mathbf{Y}N} \right) \right| > \textbf{threshold}$$

**then** Y is implicated as a black hole

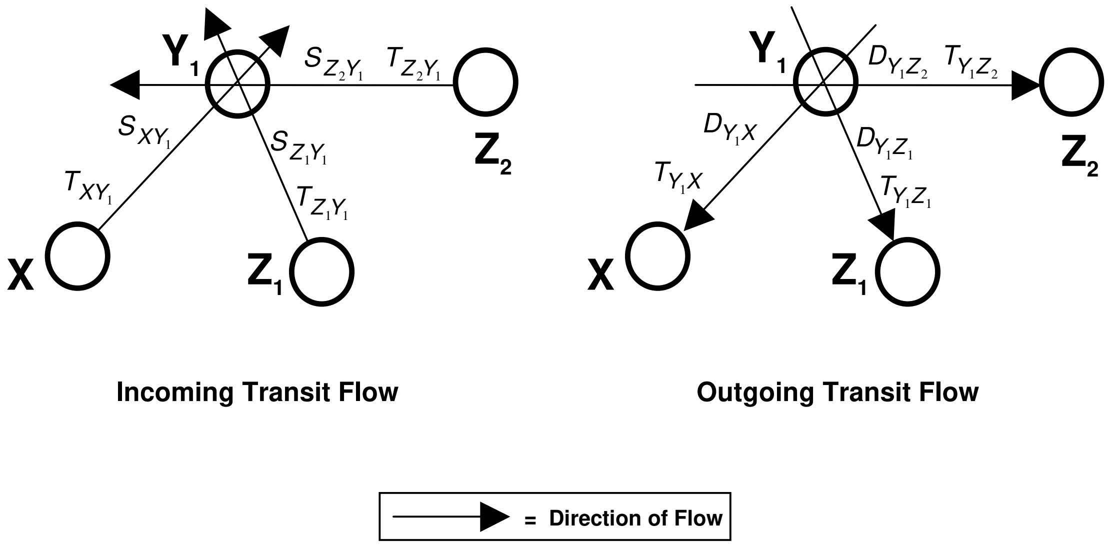



**Figure 3.12 Incoming transit flow vs. outgoing transit flow: X testing $Y_1$ from Figure 3.11.**

### 3.4.3 Overview of Diagnosis Methods

The reader may have noticed that the two methods (validation and conservation of flow) used in the diagnosis portion of WATCHERS result in two different *degrees* of diagnosis. The difference is subtle, so we clarify it here. The validation method results in individual neighbors severing their shared links to a bad router, i.e., those links whose traffic the bad router attempted to lie about. If a bad router has four neighbors and it lies about the traffic (during a particular phase, *p*) sent on only three of those four shared links to its neighbors, *then only those three shared links are severed* (during phase-*p* diagnosis). On the other hand, if a router, say *B*, has made it through validation analysis, then its counters match up with its neighbors counters, and conservation-of-flow analysis is performed on *B*.

Note that, at this point, *conservation-of-flow analysis will be performed on B by ALL of its good neighbors*! To see this, notice that each good neighbor uses the same counter values to analyze *B*, and so each neighbor will make the same decision about *B*. Conservation-of-flow analysis is different from validation analysis, in that it diagnoses the entire tested router, by evaluating all of the links incident to the tested router. Therefore, since all the neighbors use the same counters, they will all arrive at the same diagnosis, in the same phase.

We can think of these two types of diagnosis as *link diagnosis* and *node diagnosis*.

- *Link diagnosis* – the type resulting from the validation method – only diagnoses the *shared link* between the *testing router* and the *tested router*, and is used to catch *liars*. A router may lie about traffic along one or more of its links; however, each time it uses a link to lie that link will be severed in the next phase, and eventually the bad router will be removed from the network.
- *Node diagnosis* – the type resulting from the RRR sub-protocol, or conservation-of-flow analysis – diagnoses the entire node as bad, and any other router immediately neighboring that *tested node* will make the same decision (using the same counters), resulting in each good router severing its neighboring relationship with the bad router.

In short, link diagnosis of a bad router may take multiple phases of WATCHERS until all the shared links of a bad router are severed, whereas node diagnosis results in every neighbor of a bad router severing their shared link in the same phase.

Pseudo-code algorithms for the RRR and diagnosis portions of WATCHERS are provided in Figures 3.13, 3.14, and 3.15.

## 3.5 Output – Diagnoses and Reaction

The detection engine outputs link descriptors for those *shared links* that need to be severed. Severing a link requires two actions by the *testing router*. First, it floods a routing update advertising the *shared link* as down (inoperable), which has the effect of removing that link from the network map used by the other AS routers when making routing decisions. Second, the *testing router* ceases to send traffic along that link, or acknowledge traffic received on that link, which has the effect of notifying the bad router that the link is down (i.e., the testing router is not forwarding the bad router's packets). Once the *shared link* is severed, the *testing router* and *tested router* no longer have a neighboring relationship [VWW97]. Once all the links incident to a bad router are severed, the bad router is considered logically removed from the network. In Section 3.9, we prove that each bad router will eventually be removed from the network.

## 3.6 Threshold Issues

So far, we have been working under the implied assumption that good routers never have cause to drop packets. This is not a good assumption, since network protocols, including several within the TCP/IP suite, provide provisions where packets are supposed to be dropped due to errors in transmission or congestion in the network [Come95]. A limited number of packets dropped for these reasons must therefore be expected and allowed by WATCHERS. We make this type of provision using a *threshold* mechanism. Under a threshold paradigm, a router can drop $\tau$ (where $\tau$ is a *threshold value*) packets before it is considered bad. Thus $\tau$ is an upper bound on the acceptable number of packets that a router may drop before raising a flag, resulting in one or more of its links being severed.

The threshold value must also account for any timing skew between neighbors with respect to phase initiation. When a router begins a new phase of WATCHERS, it stores and sends its counters for that phase. If two good neighbors start that phase at disjoint times, their counter values may disagree even though they are good. Note, however, that this disagreement should be small, since any two good neighbors should receive a majority of respond messages at nearly the same time, and therefore initiate a new phase at nearly the same time (i.e., using flooding, the difference should be less then or equal to the delay on their shared link). Therefore, this contribution to the threshold value should be small and fairly stable between phases.

A third contributor to the need for thresholds has already been introduced in Section 3.3.2. In a dynamic network, neighbors will not continually agree on the shortest-path routes. During those times when they disagree, they may mistakenly increment their M counters, indicating misrouted packets. As we discussed in Section 3.3.2, this is not expected to happen often or for very long. Therefore, this contribution to the threshold should be small as well.

The first threshold contributor (1) only affects the transit counters (S, D, and T), since they log traffic to test for dropped packets. The second threshold contributor (2) affects all of the counters. The third threshold contributor (3) affects only the misrouting counter (M). For this reason, we will need multiple thresholds: at least two, one for transit counters [contributors (1) and (2)] and one for misrouting counters [contributors (2) and (3)].

Setting the thresholds in a practical implementation would depend on the network load and how dynamic the network topology generally is. The threshold values could change dynamically or be statically set. These threshold issues are common to intrusion detection systems such as ours and are further discussed in [SoDi95].

## 3.7 Performance and Scaling Issues

There are some properties of the WATCHERS protocol that allow enhancements to increase its efficiency. We summarize the most marked ones below.

- First, each router need only diagnose its neighbors (this is generally a small quantity) and therefore store only the counters and routing tables for its immediate neighbors, and the counters for its two-hop neighbors.
- Second, many counters will never be incremented, so if storage is an issue, counters can be stored in a linked list, greatly reducing (compared to the worst case) the amount of storage a typical router would need. In the worst case, a router stores all the counters, which equates to:

$$N \times R \times 6 + N$$

counters, where $N$ is the number of neighbors the router has and $R$ is the number of destinations in the AS. To elaborate, each router stores 6 transit counters for each neighbor, for each destination, which is $N \times R \times 6$ counters (see Figure 3.4). Furthermore, each router stores one M counter for each neighbor, which is $N$ more counters. However, in the case where an AS's topology is stable throughout a phase, each router need only store one of each counter for each neighbor, or $N \times 7$ counters ($N \times 6 + N$), because each packet going to the same destination, say $B$, will generally use the same next hop (if its routing table is stable), say $H$, and as a result will only increment counters associated to that next hop (in this case $X.S_{XH}[B]$, $X.D_{XH}[B]$, $X.T_{XH}[E]$). Since $R$ is usually much greater than $N$, the storage saved, by using a linked list including only those counters being used increases proportionally with the size of the AS, and can be significant.

- Third, incrementing the counters only calls for two additional comparison operations per packet (see Figure 3.7), which makes the cost of providing bad-router detection a small additional strain on the router's speed for processing a packet.

The first two items above also help WATCHERS scale to hierarchical network infrastructures such as in the Internet. Since (1) WATCHERS runs within an AS, (2) each router in the AS performs its own diagnosis, and (3) each router only needs to store the counters it is using (a small amount in a reasonably static network), WATCHERS resists becoming bloated as the network size grows. We discuss scaling issues further, in Section 4.2.

## 3.8 Summary of Assumptions

We make a total of five assumptions to facilitate the effectiveness of the WATCHERS router security protocol. Before we present proofs of the protocol's effectiveness, we summarize those assumptions, and how we use them.

1. *GOOD NEIGHBOR RULE: We assume that any bad router is connected by a good link to at least one good router.*
    - This assumption allows any bad router to reach a good router using a direct link, and is used in conjunction with the good path rule.
2. *GOOD PATH RULE: We assume that each pair of properly functioning routers is connected by at least one path of properly functioning links and routers.*

- This assumption, paired with the good neighbor rule, is necessary to guarantee that a message sent by a two-hop router will always arrive, regardless of whether the two-hop router is good or bad. Thus, we can assume that any message expected, but not received by a testing router, was in fact not sent, thereby labeling the expected sender as bad. We use this feature in the RRR module of the protocol, after the three stages of initiation.

3. *NEIGHBORING TOPOLOGY RULE: We assume that each pair of properly functioning routers agree on the network topology an overwhelming percentage of the time.*

   - This assumption is necessary for misroute detection. Since each router must generate and maintain a routing table for each of its neighbors, the network map which it and its neighbors use is assumed to generally be the same. We acknowledge in Section 3.3.2 that there exist acceptable cases when this does not hold, but argue that these cases are infrequent and short, and can be compensated for by using thresholds.

4. *MAJORITY GOOD RULE: We assume that good routers are in the majority.*

   - This assumption is necessary for AS-wide initiation. It allows us to preclude bad routers from falsely initiating the protocol in an attempt to slow the network. If a majority (more than 50%) of the routers in the network are good, and we only start the protocol when a majority of the routers have sent authenticated request messages, we can assume that at least one good router is involved in each initiation of the protocol.

5. *CLOCK-TICK RATE RULE: We assume that the clocks of good routers tick at equivalent rates. That is, each good router has an equivalent perception of the speed at which time passes. (This is distinct from the property of synchronized clocks. We do not require the perception of time to be equivalent, only tick rates.)*

   - This assumption is necessary for AS-wide initiation. If we assume that all good routers agree on the elapsed time since the last phase of WATCHERS began, and we know that good routers reset their phase-start counter at the start of each phase, we can assume that if all good routers have equivalent clock-tick rates, their time-out counters will run out at equivalent times, prompting local initiation. The result will be a majority of the routers (at least the good ones) sending request messages at equivalent times, causing AS-wide initiation.

## 3.9 Analysis

We use three criteria to evaluate WATCHERS as a distributed intrusion detection network monitor.

- *Correctness:* When a router is diagnosed as bad by good routers, it is in fact a bad router.
- *Effectiveness:* If a network contains any bad routers, one or more will be detected.
- *Completeness:* Every bad router will eventually be detected and removed from the network, leaving the good routers connected.

### *THEOREM 3.9*

*The WATCHERS protocol is correct, effective, and complete.*

We prove Theorem 3.9 using a lemma for each of our three criteria. We assume a zero-tolerance model (i.e., threshold = 0), for simplicity. If the threshold were greater than 0, the inequality symbols below would be replaced by threshold comparisons like those in Figure 3.10.

### *LEMMA 3.9.1*

*The WATCHERS protocol is correct.*

### *PROOF OF LEMMA 3.9.1 (by case analysis)*

*A router, say router b, can be diagnosed as bad by a good router, say router g, in three cases:*

- *(1) DIAGNOSIS BY RRR: Router g* is a neighbor of *router b*, and *router g* does not receive an authenticated respond message from *router b*. Since we assume that all good routers are connected, that each bad router has at least one good neighbor, and that WATCHERS' messages use flooding for transmission, a respond message not arriving at *router g* implies that *router b* did not send it.

- *(2) DIAGNOSIS BY VALIDATION: Router g* has already performed diagnosis *(1)* on neighboring *router b*, and finds the following result[6]:

$$\exists d\colon \left(g.T_{b,g}[d] \neq b.T_{b,g}[d] \vee g.S_{b,g}[d] \neq b.S_{b,g}[d] \vee g.D_{b,g}[d] \neq b.D_{b,g}[d]\right)$$

*or*

$$\exists d\colon \left(g.T_{g,b}[d] \neq b.T_{g,b}[d] \vee g.S_{g,b}[d] \neq b.S_{g,b}[d] \vee g.D_{g,b}[d] \neq b.D_{g,b}[d]\right)$$

Since the compared counters represent logs of the same traffic along *b* and *g*'s shared link, their inequality implies that either *router b* or *router g* has lied about the traffic that has traversed their shared link. Consequently, since *router g* is good, *router b* must be the bad router.

- *(3) DIAGNOSIS BY CONSERVATION-OF-FLOW ANALYSIS: Router g* has already performed diagnoses *(1) and (2)* on neighboring *router b*, and finds the following result:

$$\exists d\colon\left( \sum_{\forall t \in \mathrm{IN}(b)} (b.\mathrm{T}_{t,b}[d],\ b.\mathrm{S}_{t,b}[d]) \neq \sum_{\forall t \in \mathrm{OUT}(b)} (b.\mathrm{T}_{b,t}[d],\ b.\mathrm{D}_{b,t}[d]) \right)$$

This inequality between router *b*'s incoming transit traffic and outgoing transit traffic implies that *router b* is disrupting its conservation of flow with respect to transit traffic; the amount of transit traffic entering *router b* does not equal the amount exiting it. ■

[6] To reiterate, the notation $g.T_{b,g}[d]$ is read as *g's value for T counter transit traffic received from b with destination d.*

***LEMMA 3.9.2***

*The WATCHERS protocol is effective.*

***PROOF OF LEMMA 3.9.2 (by case analysis)***

*We give a proof that coincides with each of the above diagnosis types: (1), (2), and (3).*

- *(1) DIAGNOSIS BY RRR:* If *router b* attempts to be disruptive to the network by not complying with the other routers running WATCHERS it will be diagnosed – during the RRR portion of the protocol – as bad by each of its good neighbors, e.g., *router g*. Further, since we assume that each bad router has a good neighbor, *all routers* which do not comply with WATCHERS are detected by each of their good neighbors.
- *(2) DIAGNOSIS BY VALIDATION:* If *router b* attempts to be disruptive to the network, and evade detection by lying about its logs, it will be diagnosed – during the validation portion of the protocol – as bad by *one or more* of its good neighbors, e.g., possibly by router g. Further, since we assume that each bad router has a good neighbor, *all routers* which attempt to falsify their counter values are detected by *one or more* of their good neighbors (refer to Section 3.4.3).
- *(3) DIAGNOSIS BY CONSERVATION-OF-FLOW ANALYSIS:* If *router b* attempts to be disruptive to the network, and is honest about its logs, it will be diagnosed – during the conservation-of-flow analysis portion of the protocol – as bad by each of its good neighbors, e.g., *router g*. Further, since we as-

sume that each bad router has a good neighbor, *all routers* which attempt to disrupt the flow of transit traffic are detected by each of their good neighbors (refer to Section 3.4.3). ■

### ***LEMMA 3.9.3***

*The* WATCHERS *protocol is complete.*

### ***PROOF OF LEMMA 3.9.3 (by Lemmas 3.9.1 and 3.9.2)***

With Lemma 3.9.1, our good-path assumption, and our assumption that each bad router must have at least one good neighbor, we know that *only* edges between good and bad routers, or bad and bad routers, are severed. Therefore, good routers remain connected. Furthermore, by Lemma 3.9.2, we know that when a router acts maliciously, it will be detected in the next phase of WATCHERS. Each time a router is found to be bad by another router, their shared link is severed. Thus, each misbehaving router will eventually be removed from the network. ■

### ***PROOF OF THEOREM 3.9***

The proof follows immediately from Lemmas 3.9.1, 3.9.2, and 3.9.3. ■

# Chapter 4

# Conclusion

## 4.1 Results

This thesis presented WATCHERS, a distributed network monitoring intrusion detection protocol aimed at detecting denial-of-service attacks against routers. WATCHERS is designed to monitor traffic flow conservation, pinpointing any routers that disrupt that flow. WATCHERS is based on simple principles, meant to detect not only attacks but also preclude counter attacks against the protocol's own mechanisms. WATCHERS is based on the intrusion detection model, detecting and removing routers from the network automatically. It can be added to the network as a *retrofit* security mechanism, one that does not require any changes to the rest of the network infrastructure. Adding a new router to an AS running WATCHERS only requires the router to be equipped with its own copy of WATCHERS, and simple configuration. The router will be assimilated after AS-wide initiation in the first phase following its addition. Using WATCHERS does not open the routing infrastructure to any new vulnerabilities, in that it only allows a router to cut off its own neighbors (a power routing protocols already provide without WATCHERS). We presented examples of attacks and how WATCHERS counters those attacks. We described how WATCHERS might scale to a hierarchical network such as the Internet, and we gave examples of how the protocol could be efficiently implemented. Finally, we showed that WATCHERS effectively diagnoses and removes all the bad routers in a network, without false positives, eventually leaving the network secure with only good routers.

## 4.2 Future Applications of This Work

The information gathered and distributed by the WATCHERS protocol has properties that can be extended to provide another service for monitoring entire subnetworks: monitoring and evaluating the security of transit networks. It seems that, by grouping multiple routers into a *supernode* and performing the transitive closure of the member routers' counters, we can evaluate the conservation of flow for any subsection of a network. We illustrate transitive closure in Figure 4.1. The idea is for each of the routers incident to the *supernode* to exchange counters and diagnose the *supernode* as if it were a single router. This diagnosis tells us whether the *supernode*, and thus all of the routers it contains, can be treated as a good transit network.

A good transit network is one that can be trusted to forward those packets with neither source nor destination within the network. This is an important trustworthiness property of a transit network. A good diagnosis of a supernode indicates that its routers can be safely used as a transit network. A negative diagnosis of a supernode is a warning that at least one router in the supernode is bad, which can be a signal to run the full WATCHERS protocol within the individual ASs, to pinpoint the exact bad routers. With today's Internet Service Providers (ISPs) needing to rely on, as well as pay, neighboring ASs to forward traffic, a statistic such as transit network reliability will become more practical.

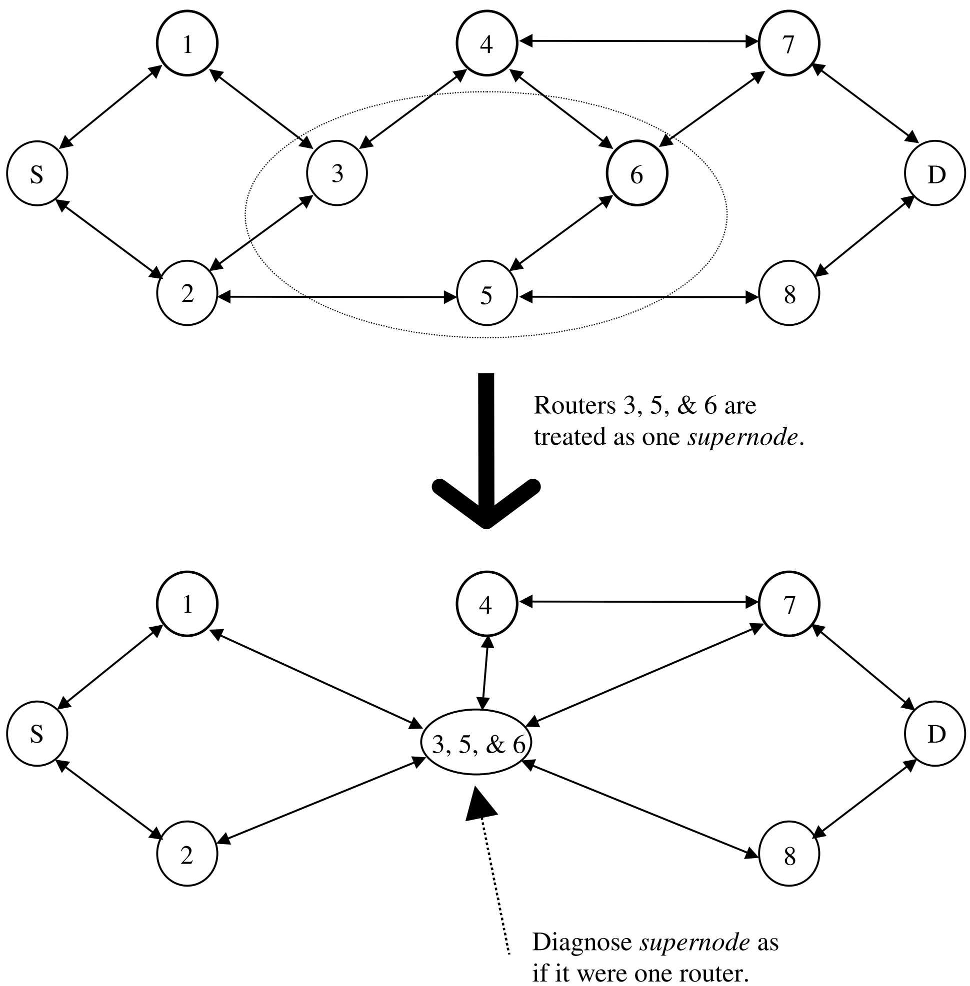



**Figure 4.1** ***Supernode*** **diagnosis of a transit network.**

Any future effort to look further into this application of WATCHERS' properties would seem to require stronger assumptions about the network. Further, when the transit network being tested contains routers from multiple ASs, the assumptions would need to apply to all of those ASs. An elegant algorithm (one that addresses the above issues) for testing transit network reliability would therefore be a valuable future application of this work.

## 4.3 Future Work

WATCHERS alone will not secure the routing infrastructure. More work needs to be done to make routing fully secure. The issues brought out in this work include:

- We employed two strong assumptions: that all good routers are connected and that each bad router has one good neighbor. These assumptions are necessary for WATCHERS' correctness. Relaxing or removing these assumptions appears to be non-trivial, but would strengthen the algorithm and constitute valuable future work.
- We do not cover the detection of all attack models. For example, a router may cause denial of service on a packet by changing one or more fields in the packet's header or data. Pinpointing attackers of this type seems to require extensive hop-by-hop integrity checking of each packet. Future work should include looking into effective ways to detect such denial-of-service attacks.
- Although we discuss performance overheads, and have tested WATCHERS under simulated malicious conditions, extensive testing of WATCHERS in a routing infrastructure under real-world conditions would be valuable future work, and should include performance diagnosis, and optimal threshold estimation (what threshold

values maximize correct, effective, and complete detection, and at the same time minimize false positives?).

# References


[ASDM94] C. Alaettinoglu, A. U. Shankar, K. Dussa-Zieger, and I. Matta. "Design and Implementation of MaRS: A Routing Testbed," Journal of Internetworking Research and Experience, vol. 5, no. 1, pp. 17-41, March 1994.

[Bell89] S. M. Bellovin, "Security Problems in the TCP/IP Protocol Suite," Computer Communication Review, vol. 19, no. 2, pp. 32-48, April 1989.

[Blac94] U. Black, Network Management Standards, Second Edition, McGraw-Hill, 1994.

[BrMa96] S. O. Bradner and A. Mankin, IPng: Internet Protocol Next Generation, Addison-Wesley, 1996.

[CFSD90] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," RFC 1157, May 1990.

[ChBe94] W. R. Cheswick and S. M. Bellovin, Firewalls and Internet Security: Repelling the Wily Hacker, Addison-Wesley, 1994.

[Cheu97] S. Cheung, "An Efficient Message Authentication Scheme for Link State Routing," To appear in Proc. 13th Annual Computer Security Applications Conference, 1997.

[ChLe97] S. Cheung and K. N. Levitt, "Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection," To appear in Proc. New Security Paradigms Workshop, 1997.

[Come95] D. E. Comer, Internetworking with TCP/IP: Principles, Protocols, and Architecture, vol. 1, Third Edition, Prentice Hall, 1995.

[Denn83] D. E. Denning, Cryptography and Data Security, Addison-Wesley, 1983.

[Denn87] D. E. Denning, "An Intrusion-Detection Model," IEEE Transactions on Software Engineering, vol. SE-13, no. 2, pp. 222-232, February 1987.

[Dick94] M. Dickie, Routing in Today's Internetworks, Van Nostrand Reinhold, 1994.

[FaWi93] D. Farmer and W. Venema, "Improving the Security of Your Site by Breaking Into It." posted on Usenet December 2, 1993. Available at: *http://www.alw.nih.gov/Security/Docs/admin-guide-to-cracking.101.html*

[Finn88] G. G. Finn, "Reducing the Vulnerability of Dynamic Computer Networks," ISI Research Report ISI/RR-88-201, University of Southern California, Marina del Rey, California, June 1988.

[Fris95] A. Frisch, Essential System Administration, Second Edition, O'Reilly & Associates, Inc., 1995.

[GaSp96] S. Garfinkel, and G. Spafford, Practical UNIX and Internet Security, Second Edition, O'Reilly & Associates, Inc., 1996.

[Hedr88] C. Hedrick, "Routing Information Protocol," RFC 1058, Rutgers University, June 1988.

[Held96] G. Held, LAN Management with SNMP and RMON, John Wiley & Sons, 1996.

[Hind96] R. M. Hinden, "IP Next Generation: Overview," Communications of the ACM, vol. 39, no. 6, pp. 61-71, June 1996.

[HPT97] R. Hauser, T. Przygienda, and G. Tsudik, "Reducing the Cost of Security in Link-State Routing," Proceedings of the Symposium on Network and Distributed System Security (SNDSS), San Diego, California, pp. 93-99, February 1997.

[Huit95] C. Huitema, Routing in the Internet, Prentice Hall, 1995.

[IoBl93] J. Ioannidis and M. Blaze, "The Architecture and Implementation of Network-Layer Security under UNIX," Proceedings of the 4th Usenix Security Workshop, Santa Clara, October 1993.

[KuCr93] B. Kumar, and Jon Crowcroft, "Integrating Security in Inter-Domain Routing Protocols," Computer Communication Review, vol. 23, no. 5, pp. 36-51, October 1993.

[Kuma93] B. Kumar, "Integration of Security in Network Routing Protocols," ACM SIGSAC Review, vol. 11, no. 2, pp. 18-25, Spring 1993.

[Malk93] G. Malkin, "RIP Version 2 - Carrying Additional Information," RFC 1388, Xylogics, Inc., January 1993.

[MRR80] J. M. McQuillan, I. Richer, and E. C. Rosen, "The New Routing Algorithm for the ARPANET," IEEE Transactions on Communications, vol. COM-28, no. 5, pp. 711-719, May 1980.

[MHL94] B. Mukherjee, L. T. Heberlein, K. N. Levitt, "Network Intrusion Detection," IEEE Network, vol. 8, no. 3, pp. 26-41, May-June 1994.

[Moy94] J. Moy, "OSPF Version 2," RFC 1583, March 1994.

[MuBa96] S. L. Murphy, M. R. Badger, "Digital Signature Protection of the OSPF Routing Protocol," Proc. of the Symposium on Network and Distributed System Security (SNDSS), pp. 93-102, 1996.

[Pace97] J. E. Pace, Malicious Router Detection Through Router Audit, Thesis: Master of Science, University of California, Davis, 1997 (in preparation).

[Perl88] R. Perlman, Network Layer Protocols with Byzantine Robustness, PhD Dissertation, Massachusetts Institute of Technology, October 1988.

[Perl92] R. Perlman, Interconnections: Bridges and Routers, Addison-Wesley, 1992.

[PWMO97] J. Pace, C. Wee, B. Mukherjee, and R. Olsson, "Malicious Router Detection," To appear in Proc. 1997 UC Davis Student Workshop on Computing, University of California, Davis, 1997.

[PZCMO96] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. Olsson, "A Methodology for Testing Intrusion Detection Systems," IEEE Transactions on Software Engineering, vol. 22, no. 10, pp. 719-729, October 1996.

[SiKe97] K. E. Sirois and S. T. Kent, "Securing the Nimrod Routing Architecture," Proceedings of the Symposium on Network and Distributed System Security (SNDSS), San Diego, California, pp. 74-84, February 1997.

[SoDi95] B. Soh and T. Dillon, "Setting Optimal Intrusion-detection Thresholds," Computers and Security, vol. 14, no. 7, pp. 621-31, 1995.

[SMG97] B. R. Smith, S. Murthy, and J. J. Garcia-Luna-Aceves, "Securing Distance-Vector Routing Protocols," Proceedings of the Symposium on Network and Distributed System Security (SNDSS), San Diego, California, February 1997.

[Stee95] M. Steenstrup "Routing in Communications Networks" Prentice Hall, 1995.

[VWW97] B. Vetter, F. Wang, and S. F. Wu "An Experimental Study of Insider Attacks for the OSPF Routing Protocol," Proc. International Conference on Network Protocols (ICNP '97), Atlanta, GA, October 1997.

[WPB97] C. Wee, J. E. Pace, and K. A. Bradley "Routers and Network Auditing," Technical Report (in preparation), Dept. of Computer Science, University of California, Davis, 1997.