Conservation of Flow as a Security Mechanism in Network Protocols

By

JOHN ROBERT HUGHES
B.S. (Purdue University) 1998

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. Matt Bishop, Chair*

_____ *Dr. Karl Levitt*

_____ *Dr. Biswanath Mukherjee*

Committee in Charge

2000

Acknowledgments

I wish to thank my Committee members, Matt Bishop, Karl Levitt, and Biswanath Mukherjee, who have provided valuable feedback during this endeavor. Special acknowledgement is due to Tuomas Aura and Matt Bishop, who co-authored with me the paper upon which this thesis expands [HAB00].

This research was sponsored by grant NAG21251 from the National Aeronautics and Space Administration and a gift from Microsoft Corporation to the University of California, Davis.

I wanted to thank
Everyone I know.
From their contributions
This thesis did grow:

My mother, my father,
And Carol as well,
But not my neighbors' dogs,
Which made my life Hell.

Only Matt and Karl,
Biswanath and Nick,
Steven and Dean,
And Tuomas I pick,

As those most helpful,
With their wise insight.
The list ends here,
But you might delight:

For I wanted to thank
Everyone I know,
But both my Committee
And printing office said No.

Table of Contents

1 Introduction

The law of conservation of flow states that an input to a system must either be absorbed by that system, or be sent on as an output, possibly with modification. For an ideal system, this law should hold, for no messages should be dropped in transit, nor should any source or destination nodes falsely repudiate having sent or received messages, respectively. Unfortunately, when the idealized law of conservation of flow is applied to a realistic situation, *e.g.*, Internet routing, various considerations must be made in order to compensate for the model's inadequacy. This thesis considers the application and implementation of WATCHERS [Brad97, BCPM+98a, BCPM+98b], an existing protocol designed to detect misbehaving routers using the law of conservation of flow.

1.1 Overview

Conservation of flow is an attractive tool to analyze network protocols for security properties. One of its uses is to detect disruptive network elements that launch denial of service (DoS) attacks by absorbing or discarding packets. The WATCHERS protocol describes a network of coordinated, distributed network monitors, each of which apply *validation* and *conservation-of-flow* tests to their neighbors. This application requires many assumptions about the protocols and networks being analyzed. Unfortunately, WATCHERS' implicit assumptions do not hold in the Internet. The protocol can consequently be defeated.

1.2 Motivation

In a word, **money**.

The Internet's explosive growth has introduced unprecedented opportunities for communication and information sharing. As the online population grows, and electronic commerce matures, our economy may soon depend on this worldwide network. At the heart of it all are *routers*. Unfortunately, routers are vulnerable.

Just as routers can be used passively in an attack, they too can become the active element, should an attacker gain control of them. Although relatively few such router vulnerabilities have been widely exploited, if and when the currently "popular attacks" are prevented or lose their appeal, attackers will almost certainly seek out new vulnerabilities, including those in routers and their control mechanisms.

Should a significant weakness be discovered and exploited, depending on the vendors' response time, an attack on the very infrastructure of today's Internet could certainly affect the infrastructure of tomorrow's economy [Seat00a]. Recent attacks have even had a significant, perhaps intentional, effect on the U.S. stock market [Merc00, Seat00b, Wash00]. In one survey, 273 participating organizations reported annual losses of $265 Billion due to computer security breaches [CSI00].

Regardless of their motive, attackers who control routers pose a significant threat. While it may be difficult to foresee the weaknesses they might exploit to gain such power, vulnerabilities are almost certain to exist. The persistence of well-understood types of vulnerabilities such as buffer overflows is evidence that even when prevention methods are known and available, it does not guarantee that these weaknesses will be eliminated [Vene96]. With respect to router vulnerabilities, the potential damage an attacker might cause can be limited by automatic detection of the attack and subsequent

isolation of the offending router(s) from the network. WATCHERS is a distributed network monitoring protocol designed to accomplish this.

1.3 Objectives and Contributions

This thesis explores the WATCHERS protocol, its deficiencies, and how it can be improved. The primary benefit is an implementation of WATCHERS. This was used to gather empirical results in the form of performance data and proof-of-concept evidence, of both WATCHERS' usefulness and weaknesses.

In the course of analyzing the WATCHERS protocol, many attack scenarios are presented, each of which poses a particular threat to WATCHERS' accuracy or robustness. Some scenarios target the use of conservation of flow, while others manipulate the supporting routing protocol and network transport features. This exploration resulted in numerous suggested improvements to allow WATCHERS to function in the presence of malicious routers that might attempt to disrupt the very protocol designed to detect them.

Even with these improvements, many more questions remain unanswered. Despite its outward simplicity, WATCHERS has many parameters and provides intrinsic opportunities for further research. Many of these possibilities exist only now that an implementation has been created.

1.4 Thesis Organization

This document is partitioned into five main chapters. This first chapter has introduced to the reader the concept of conservation of flow and its uses in computer security protocols today. The motivation, objectives, and contributions of this work have

also been discussed. The next chapter will expand the motivation by providing a background to Internet routing and an overview of related work. If the reader is familiar with the Internet, routers, and routing protocols, and not interested in a brief history of WATCHERS' evolution, feel free to skip to chapter 3, where a concise presentation of the WATCHERS protocol is presented, as it existed prior to the publication of this thesis. If the reader is intimately familiar with WATCHERS, it may be most appropriate to begin in section 3.3, where attacks on the protocol are introduced.

In chapter 4, a WATCHERS implementation is presented, along with experimental results related to the observations in chapter 3. Finally, closing remarks are found in chapter 5, including a summary of the results obtained and directions for future research.

2 Background

This chapter provides a brief background of routers, routing protocols, and their historical vulnerabilities. Section 2.4 reviews the evolution of WATCHERS and related work. It is assumed that the reader has a basic understanding of IP, *traceroute*, *ping*, *telnet*, and *ftp* [Come95].

2.1 Routers

Routers are machines that direct traffic flow on any sizeable computer network: Every packet of data a router receives must be correctly forwarded to the next appropriate router, or *hop*. In order to do this, each router maintains a *routing table* listing the appropriate next hop for specific destinations and/or destination networks. Upon receiving a packet, a router parses that packet's *header*, extracting, among other things, the destination address. The router then checks its routing table, performs any maintenance or special instructions requested in the packet header, and sends the packet out on the correct network interface.

2.2 Routing Protocols

It is important that the routing tables be kept current in order to prevent misrouting and network inefficiencies. The task of synchronizing routing information is delegated to *routing protocols*. Prevalent Internet routing protocols include the vector-distance Routing Information Protocol (RIP) [Malk98] and the link-state Open Shortest Path First (OSPF) [Moy98] protocol [Come95, Huit00].

The difference between vector-distance and link-state routing protocols lies in the information exchanged between neighboring routers. Vector-distance protocols propagate a list of reachable networks, and the distance to each. Link-state protocols propagate the status of each individual connection. In order to determine distance and reachability, a router running a link-state routing protocol performs a local computation based on the "up" or "down" status of each link.

The finer granularity offered by link-state routing protocols is necessary for a misbehavior detection protocol such as WATCHERS. In order to oversee neighboring routers' proper compliance with the WATCHERS protocol, it must know whether each link is up or down. This information is particularly necessary in order to verify that a neighbor has not misrouted packets, in addition to other requirements (See section 3.2.2). Additionally, OSPF can be configured to authenticate routing messages, making it the recommended routing protocol to complement WATCHERS. (It should be noted that even in OSPFv2, the built-in cryptographic authentication scheme relies on a shared secret key, and thus in inadequate to prevent identity spoofing; public-key digital signatures are one solution, but are computationally expensive. Some alternatives are discussed in [Cheu97, Zhan98].)

2.3 Attacks

Routers and routing protocols have had their share of weaknesses, both in their design and implementation. To date, a least one major networking vendor averages more than two serious vulnerabilities each year [Cisc00], where a serious vulnerability is defined as an opportunity for an attacker to gain control of the router or its packet forwarding rules.

Compromised routers may cause all packets to be dropped, constituting a *network sink* (or a *black-hole router* if it also falsely advertises zero- or low-cost routes). Malicious routers can also selectively drop packets, constituting *intelligent black-holes*. Routers may misroute packets or even conspire to misbehave together, the latter constituting *consorting routers*. The WATCHERS protocol was designed to detect and isolate routers participating in these malicious activities.

2.4 Previous and Related Work

The concept of information flow is not new; security-related models have been developed from everything from propagation of user rights [BeLa73] to the flow of "insecurity" in vulnerable systems [MoKa97]. Sampling and visualization of network flows is discussed in [FSPS95]. Some recent intrusion detection systems attempt to identify individual network flows for the purposes of tracing attackers [SCCD+96]. WATCHERS attempts to discover malicious network elements that improperly route network flows.

2.4.1 Conservation of Flow

To this author's knowledge, use of conservation of flow as a security mechanism in network protocols is a relatively new and unexplored idea. Cheung and Levitt introduced *Flow analysis* as a means of detecting misbehaving routers – those that either intentionally or accidentally misroute or drop packets [ChLe97]. Those authors made three assumptions: (1) Neighboring routers share the same view of the network topology, (2) Routers send packets along the shortest route to their destination, and (3) Neighboring

routers share bi-directional links over which they can exchange packets. Pseudo-code was also presented for router synchronization and detection of misbehavior.

These steps were formalized by Bradley with the introduction of the WATCHERS protocol: Watching for Anomalies in Transit Conversation: a Heuristic for Ensuring Router Security [Brad97]. Bradley identified additional requirements including the *good neighbor*, *good path*, and *majority good conditions* (see section 3.1). He also recognized the need for clock synchronization among participating routers.

Minor improvements to the WATCHERS protocol were presented in [BCPM+98a, BCPM+98b], including the *link-state condition* (see section 3.1), detection of certain misrouting and *consorting router* misbehavior, and detection of routers that fail to properly diagnose bad routers. The authors also explored the issues of WATCHERS' costs and misbehavior thresholds in more detail. Chapter 3 describes the current status of the WATCHERS protocol.

2.4.2 Flow Control and Quality of Service

The related concepts of *Quality of service* (QoS), *Flow Control*, and *resource reservation* have garnered much attention in the face of burgeoning computer networks. Each of these is similar to WATCHERS in that stateful network flow information is usually maintained. QoS mechanisms such as RSVP [BZBH+97] are later shown to be complementary to WATCHERS (see section 5.2.2). [BMR97] describes a generic flow measurement architecture that might meet WATCHERS' needs.

2.4.3 Byzantine Agreement

WATCHERS relies on good routers' shared counters being in agreement (approximate agreement, if non-zero thresholds are employed). Unfortunately, in the model that WATCHERS requires (see section 3.1), each link connects only two routers, and as such, only these two entities have firsthand knowledge of what traffic actually passes over their shared link. Since Byzantine agreement requires at least three well-behaved parties to reach agreement in the presence of just one malicious party, solutions to the Byzantine agreement problem will not be of assistance to counter agreement where a bad router is involved. If two good routers disagree on shared counter values, the problem is not that one of them is lying, but instead that each of them truly believes their asserted values are correct; this incongruity cannot be resolved by an agreement protocol, and a solution must instead focus on eliminating the underlying cause(s) of the counter discrepancy.

One context in which Byzantine agreement might be applied to WATCHERS relates to the participating routers' decision on when to start a new WATCHERS round (See section 3.2.1). Currently, WATCHERS requires only a majority consensus, relying on several required *conditions* to loosen the restrictions that would otherwise be necessary if a solution to the Byzantine or other agreement problem were used. Specifically, WATCHERS requires that all messages be authenticated and *flooded*, a path of good routers connects each pair of good routers, and a majority of routers in the system are good (see section 3.1). Provided that connections are loss-less, these *conditions* effectively guarantee receipt of good routers' messages by all other good routers, as would be the case in a fully connected, loss-less network. In such a situation,

only a majority of good routers is required, and again, WATCHERS has no need to rely on a more restrictive agreement protocol.

3 The WATCHERS Protocol

In this chapter, the WATCHERS protocol and its model are reviewed. Associated terminology and required *conditions* are presented first, followed by an overview of the RRR sub-protocol, the *validation* and *conservation-of-flow* tests, and WATCHERS' response mechanism. Weaknesses in the protocol and its model are then explored using various attack scenarios. Finally, a summary and analysis is given, exploring the ramifications of the weaknesses presented.

3.1 Model and Terminology

WATCHERS, first introduced in [Brad97] and later improved upon [BCPM+98a, BCPM+98b], is a distributed network monitoring protocol designed to detect and isolate malicious routers within an autonomous system (AS). For WATCHERS' purposes, a bad, or *malicious router* is defined as one that discards or misroutes (sub-optimally routes) packets, or does not participate in or gives incorrect information during execution of the protocol. Except where noted, the most recent version of WATCHERS is described [BCPM+98b].

In order to function correctly, WATCHERS requires four conditions:

1. *Link-State Condition*: A link-state routing protocol must be used, where each router is aware of all routers and links between them within the AS. Each router periodically broadcasts an update message to inform the other routers the "up" or "down" status of each of its links.
2. *Good Neighbor Condition*: Every router must be directly connected to at least one non-malicious router.
3. *Good Path Condition*: A path of good routers must connect each pair of good routers.
4. *Majority Good Condition*: There must be more good routers than bad.

Figure 3.1: Conditions required for WATCHERS to function correctly.

Additionally, it has been claimed [BCPM+98a, BCPM+98b] that WATCHERS is *correct, i.e.,* a good never diagnoses another good router as bad, provided the following two *conditions* hold:

1. *Perfect Transmission Condition*: When any router sends a WATCHERS message to a neighbor, the message arrives intact with no delay.
2. *Neighbor Agreement Condition*: Neighboring routers *always* agree on the network topology.

Figure 3.2: Additional conditions claimed to prevent good routers from diagnosing other good routers as bad.

WATCHERS requires each pair of directly connected routers, or *neighbors,* to have only a single *shared link*. Each router maintains 7 base counters with respect to each shared link. For neighbors *X* and *Y*, counter $T_{X,Y}$ refers to data transiting through *X* and then *Y*, $S_{X,Y}$ for data originating at *X* and then sent through *Y*, and $D_{X,Y}$ for data destined for *Y* arriving through *X*. These three counters represent *incoming flow* to *Y*, but *outgoing flow* to *X*. $M_{X,Y}$ records the number of misrouted packets *X* sends through *Y*. Respective counters with "$_{Y,X}$" subscripts are maintained for data flowing in the opposite direction. Both *X* and *Y* maintain their own copies of the six counters shown in Figure 3.3, while only *X* maintains $M_{Y,X}$ and only *Y* maintains $M_{X,Y}$. Note that no counters exist for data originating with *X* and destined for *Y*, or for data exchanged between *border routers* and *external* nodes (external nodes include workstations, terminals, or any machine or router not participating in the same WATCHERS system). WATCHERS ignores such packets.

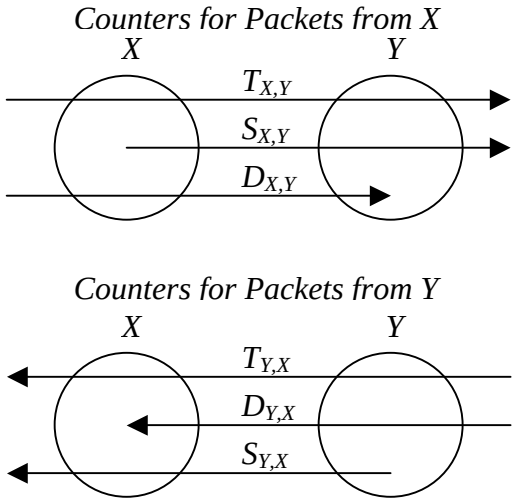

Figure 3.3: Base transit packet byte counters.[1]

While the base counters introduced thus far facilitate detection of simple misbehavior, they are insufficient to discover certain *consorting routers*, or those that cooperate to conceal their wrongdoing. In order to do so, WATCHERS logging requirements are increased: Instead of two *S* and two *T* counters per neighbor, each router must maintain two *S* counters and two *T* counters *per destination*. Thus, $T_{X,Y}[Z]$ would refer to data sent through *X* and then *Y*, destined for *Z*.

3.2 Communication in the WATCHERS Protocol

Participants in the WATCHERS protocol identify malicious routers by periodically performing *validation* and *conservation-of-flow* tests on each neighbor. In each period, or *round*, the *T*, *S*, and *D* counters must be exchanged, authenticated, and analyzed. All such Administrative messages (those sent either by WATCHERS or the routing protocol) must be *flooded*, *i.e.*, each *new message* (one not seen before) received

[1] [HAG00] mistakenly assumed that *T*, *S*, and *D* counters were intended to count *packets*. However, only *M* counters actually measure *packets*; *T*, *S*, and *D* counters measure *bytes*. Under certain conditions, counting *packets* may be most appropriate (see section 5.2.4).

by a good router must be forwarded to each of its other neighbors. Flooding, along with the *good path condition*, ensures all good routers' messages will reach every other good router in the AS. Note that because flooded messages are intermediately originated and destined for adjacent routers, these messages are not reflected in WATCHERS' counter values. Furthermore, every WATCHERS message must be digitally signed, so that the receiving nodes may authenticate the message's source and its contents.

3.2.1 The RRR Sub-Protocol

WATCHERS requires that all routers be synchronized to begin each round. The RRR sub-protocol accomplishes this in three stages: Request, Receive, and Respond. When a router is ready to begin a new round (usually because its clock indicates it is time to do so) it will flood a *request* message to all its neighbors. Upon receipt of requests from a majority of the other routers in the AS, it will send the *request* message if it has not already done so, and will then take a "snapshot" of its own counters. This snapshot is then flooded as a *response* message. When a router has received all the responses it requires (snapshots from all its neighbors and all *their* neighbors), it begins the *diagnosis* phase.

3.2.2 Diagnosis

The first test in the diagnosis phase is *validation*. For each link, *local validation* checks that the *testing router*'s counters differ by not more than a certain *threshold* (possibly zero) when compared to the corresponding counters maintained by the neighbor across the link, the *tested router*. Should a neighbor fail local validation, it is identified as bad, and further tests on it are unnecessary. A testing router performs *Remote*

validation by applying the same test between each neighbor and each of their neighbors. Should a neighbor fail remote validation, it and the neighbors with whom its shared counters disagree are added to a *check set*.[2] If at the start of the next round's diagnosis, the neighbors in the check set have not removed the shared links between themselves and the neighbors with whom their counters disagreed, such routers are identified as bad.

Validation will detect routers that change their counters, but not those that simply drop packets. The latter are detected by the *conservation-of-flow* test. A router will perform this test on each of its neighbors that pass the *validation* test. Calculated separately for each possible destination, if the difference between the tested router's *incoming transit flow* (incoming *S* and *T* counters) and its *outgoing transit flow* (outgoing *D* and *T* counters) is greater than a certain threshold, the tested router is identified as bad. Note that incoming *D* counters and outgoing *S* counters are not included in incoming transit flow and outgoing transit flow, respectively, because these do not count bytes *transiting* through the tested router.

3.2.3 Response

Once a neighbor is identified as bad, a routing update is flooded, advertising the shared link between the testing and bad routers as down, thereby signaling the other routers to also remove those links from their routing tables. WATCHERS assumes that the routing protocol being used will treat a link as down if it receives conflicting reports of its operable status. To accomplish isolation of bad routers, messages are no longer sent to, or accepted from, detected malicious routers. Once a bad router's neighbors all

[2] [HAG00] mistakenly assumed that a router would immediately diagnosed as bad if its counters disagreed with any of its neighbors. If this were the case, attacks such as Kamikaze routers would be possible [HAG00].

take such action, the bad router is *logically removed* from the network. It should be noted that this reactive approach might invalidate one or more of the required *conditions*, unbeknownst to the remaining good routers.

3.3 Attack Scenarios

Weaknesses in the WATCHERS protocol and model are explored in the attack scenarios that follow. In several scenarios, a table of WATCHERS counters is provided to illustrate the non-zero counter values following the attack. It is assumed that no other packets are being exchanged, *i.e.*, if the attack had not taken place, all counters would have a value of zero. Furthermore, for the purposes of identifying a bad router, the counter disagreement and misrouting threshold is assumed to be zero unless otherwise specified (a difference of just one misrouted packet or dropped/surplus byte indicates a bad router).

3.3.1 Packet Modification

Conservation of flow does not say *which* packets or contents proceed to the destination. It merely ensures that *some* set of packets with an equal number of payload bytes arrive at the destination. Even with *per-destination* counters, malicious routers may continue to misroute packets undetected [BCPM+98b]. As an example, suppose router *A* in Figure 3.4 sends two identically sized packets ($|P|$ denotes the number of bytes in packet *P*), one to *C* and one to *X*. Bad router *B* deliberately swaps the destination addresses: Packet P_{AX} (originated by router *A*, and intended for *X*) arrives at *C* and packet P_{AC} arrives at *X*. While the operating system or applications running on routers *C* and *X* may realize something has happened, regardless of any added nodes or links to this

topology (in satisfaction of the required *conditions*), no router will be able to detect *B*'s misbehavior using the WATCHERS algorithm.

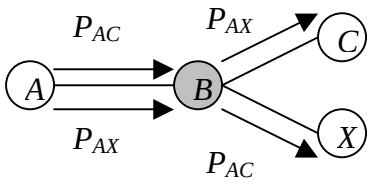

Figure 3.4: Bad router *B* Switching packets' destination addresses. For clarity, packet labels reflect the original intended destination.

A	***B***	***C***	***X***
$S_{A,B}[C]=\|P\|$	$S_{A,B}[C]=\|P\|$	$D_{B,C}=\|P\|$	$D_{B,X}=\|P\|$
$S_{A,B}[X]=\|P\|$	$S_{A,B}[X]=\|P\|$		
	$D_{B,C}=\|P\|$		
	$D_{B,X}=\|P\|$		

Table 3.1: Non-zero WATCHERS counters at the conclusion of this attack; $|P|=|P_{AC}|=|P_{AX}|$.

Note that it would be trivial for a malicious router to grow or shrink transient packets' payloads, drop and inject spoofed packets, or a combination of the two, so long as the net change to the sum of bytes in the packets is zero.

3.3.2 Packet Substitution

While *per-destination* counters have been shown to enable detection of at least one class of *consorting router* attack [Brad97, BCPM+98a, BCPM+98b], they are not sufficient to detect other classes of consorting router attacks. As with packet modification, conservation of flow does not inhibit a router's ability to substitute packets. In Figure 3.5, router *A* sends a packet, P_{AX}, destined for router *X*. Bad routers *B* and *C* conspire to drop that message, replacing it with identically sized packet P_{BX}. Routers *B* and *C* then lie by incrementing their $T_{B,C}[X]$ counters (instead of their S counters). Even when an additional path of good routers exists between *A* and *X* (satisfying the required *conditions*), as far as the WATCHERS protocol is concerned, the good routers cannot detect this misbehavior. The ability to mount this attack is unrelated to the position of the pair of bad routers in the path; unlike the next-to-last consorting router attack

[BCPM+98b], no additional good routers insulating *A* and *X* from the bad routers would affect this attack's potential.

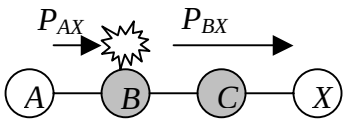

Figure 3.5: Packet Substitution.

A	***B***	***C***	***X***
$S_{A,B}[X]=\|P\|$	$S_{A,B}[X]=\|P\|$	$T_{B,C}[X]=\|P\|$	$D_{C,X}=\|P\|$
	$T_{B,C}[X]=\|P\|$	$D_{C,X}=\|P\|$	

Table 3.2: Non-zero WATCHERS counters at the conclusion of this attack; $|P|=|P_{AX}|=|P_{BX}|$.

3.3.3 Ghost Routers

Conservation of flow says nothing about the configuration of nodes over which the flow is measured. If incoming and outgoing flows are measured, the entity between the measuring points may be one node or multiple nodes; the measurer cannot tell. Indeed, this may be considered a *feature*, as [Brad97, BCPM+98a, BCPM+98b] propose the notion of a *supernode*, so as to reduce the logging requirements and improve scalability in large networks for which the topology is fully known. However, if routers are not only able to broadcast link-state network status messages, but also topological information, the following attack becomes possible. Figure 3.6 depicts how bad router *A* can announce to the network that it is really composed of two routers, *A* and *B*. The creation of such *ghost routers* allows *A* to misbehave, while shifting blame to its *ghost*(*s*). Generalizing this scenario, bad routers may invent arbitrary network topologies in place of themselves. As one example application, a single router can mount the packet substitution attack of section 3.3.2 by pretending to be two adjacent routers. Any malicious router capable of creating *ghost routers* may invalidate the necessary *majority good condition*.

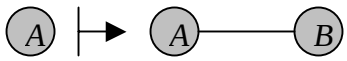

Figure 3.6: Ghost router creation.

3.3.4 Premature Aging

Internet packets have a *time to live* (TTL) field that, upon reaching 0, will cause the packet to be discarded. Even if secure transport and routing protocols are used, packets can still be prematurely aged. A router can set the TTL to 1 in both originating and transient packets, forcing the next hop to drop the packets if that router is not the packets' destination. Generalizing this, any successive router along the path could be forced to drop the packet should a malicious router set the TTL to a value less than the remaining distance to the destination.

Similar attacks are possible against link-state routing protocols. Use of link-state protocols is required by the *link-state condition*. WATCHERS' suggested routing protocol is OSPF [Moy98]; unfortunately even OSPF v2 contains vulnerable fields in its topology-advertising broadcast packets [QVWN$^+$98]. When these topology packets, or *link state advertisements* (LSAs), are passed among routers in an AS, the Age field is incremented. Upon reaching MaxAge, associated topology information is no longer used in calculating the routing table. At that time, the LSA is re-flooded across the network with its Age set to MaxAge, forcing the originating router to increment the LSA's sequence number and try to resend.

Premature aging is a necessary *feature* in OSPF. However, by setting the Age field to MaxAge for all transient LSAs, a malicious router can force frequent

retransmissions, potentially saturating the network. (Note that the Age field is the only one excluded from OSPF message checksums. Like TTL, OSPF's Age filed is dynamic, and not protected by the available message integrity schemes.) Due to the rapid topology announcements, routers are also more likely to have inconsistent views of the network topology at any given time, possibly with sections of the network being unreachable.

3.3.5 Simultaneous Exchange

A critical aspect to WATCHERS' accuracy is *timing*. If packets can be exchanged simultaneously, numerous problems may arise regarding counter disagreement, especially if counter snapshots aren't synchronized (even if they were synchronized, packets still in transit would appear to have been dropped in the *conservation-of-flow* test).

WATCHERS maintains routing tables for each of its neighbors. If updates to these local routing tables are not made until that neighbor acknowledges the LSA, malicious routers may refuse or delay their acceptance, resulting in discrepant views of the network topology. Alternatively, if updates are made prior to a neighbor's acknowledgement, or if that neighbor's outbound traffic is not sent in a strict first-in first-out (FIFO) manner, a malicious router may use false LSAs to induce packet misrouting.

In the example that follows, bad router *A* accomplishes such an attack by arranging simultaneous packet exchanges between the attacked node, *X*, and *X*'s neighbors, while the parties to the exchanges share differing views of the network topology.

Figure 3.7 (a) shows router *A* sending a broadcast topology message throughout the network indicating link *A*-*E* is down (broadcast packets are not currently accounted

for in the WATCHERS protocol, and thus have no effect on the WATCHERS counters in Table 3.3). For step-wise clarity, Figure 3.7 (b) and Figure 3.7 (c) depict shorter time slices in the message passing. Figure 3.7 (b) shows router *C* receiving *A*'s next message, claiming *A*-*E* is now up. In Figure 3.7 (c), router *C* passes this message on, simultaneously receiving P_{XA}. Even if router *E* has insisted link *A*-*E* is up, only now would router *C* believe so; *C* also concludes that router *X* has misrouted P_{XA}, the packet intended for *A* (since link *A*-*E* is now up, router *X* should be sending all traffic destined for *A* through *E*). We can thus convince the *X*'s neighbor closest to *A* that *X* is bad.

To convince the other neighbor, bad router *A* can create *ghost routers*, as described in section 3.3.3. Using such additional nodes, router *A* can arbitrarily make the clockwise or counter-clockwise path between it and any other router the shorter of the two. Thus, router *A* can use the above attack on any router in the ring from either direction, and convince both neighbors of each attacked router to consider it bad. It follows that for WATCHERS rounds of sufficient duration, a single malicious router can cause all other routers in a ring network to be falsely convicted as being bad.

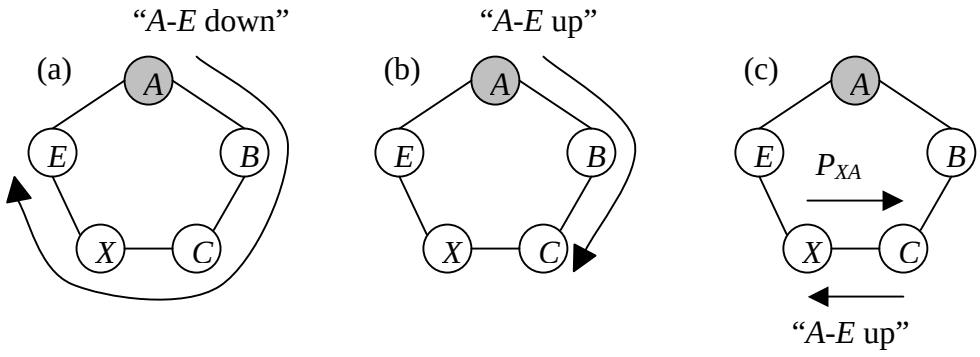

Figure 3.7: Topology changes used to induce packet misrouting.

A	*B*	*C*	*X*	*E*
		$S_{X,C}[A]=\|P_{XA}\|$	$S_{X,C}[A]=\|P_{XA}\|$ $M_{X,C}=1$	

Table 3.3: Non-zero WATCHERS counters at the conclusion of this attack.

Note that traffic can be solicited from the nodes being attacked by using acknowledgement-required protocols, and with appropriate timing, increase the likelihood of successfully exploiting this race condition. While it may not be possible for multiple messages to co-exist on a particular media, *e.g.*, Ethernet, for the purposes of achieving simultaneous exchange, the only requirement is that the communication not be serialized.

Additionally, for a ring network, it should be clear that any malicious router can remove its two neighboring routers from the ring without the use of *ghost routers*, since it can force its own shared link down after accomplishing the above attack on the neighbors' opposite link. Thus, for a ring network with only three nodes, if even one is corrupt, it can, without creating *ghost routers*, compel the other two nodes to discontinue communication. Finally, when two or more neighboring bad routers coexist in a ring network of *any* size, they can conspire to accomplish this attack, since at least one of them will always be along the shortest route to every good router on the ring.

3.3.6 Hot Potato

Even when conservation of flow holds, entities may engage in malicious activity undetected. For this attack, it is hypothesized that some commercial routers are more interested in processing packets quickly, rather than checking special and rare

conditions.[3] In a ring-network such as that in Figure 3.8, if routers are willing to believe a neighbor's claim that a shared link is down, when that message itself comes over that *same shared link*, the following attack becomes possible:

If bad router *A* continuously broadcasts the topology update messages as shown in Figure 3.8, any packet destined for *A* may be sent back and forth between *B* and *C*, due to the "thrashing" topology. During this period, the TTL of this packet may expire. Although an ICMP TTL Expired message may be sent back to the originator, WATCHERS does not view this kind of effort as compensation for the dropped packet. Thus, the router that dropped the expired packet will fail the *conservation-of-flow* test, causing it to be labeled as bad.

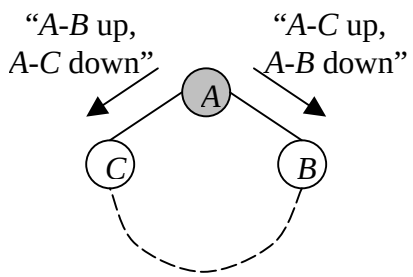

Figure 3.8: Topology changes used to delay incoming messages.

It should be noted that OSPF does ignore LSAs received within MinLSArrival seconds of one another. The above attack still might be possible if the previous LSAs sent by bad router *A* can be flushed rapidly. To accomplish flushing, *A* would prematurely age the LSA to MaxAge (see section 3.3.4), and after receiving acknowledgement from its neighbor, would then be free to send a new LSA shortly: Since the MaxAge LSA will have been removed by the neighbor when it has received acknowledgement from all of *its* neighbors, for a ring network with a sufficient number

[3] [Huit98] describes how some vendors have chosen to boost their routers' performance by not verifying IP header checksums.

of routers, such acknowledgements may be returned before the MaxAge LSA traverses the length of the network. Such replies are sent as *delayed acknowledgements*, but if this delay is sufficiently short, the preceding attack can still be effective. Since links described by a prematurely aged LSA are not considered when updating the routing table, simply sending a MaxAge LSA is, at least in this context, functionally equivalent to a message indicating a link is down.

3.3.7 Source Routing

Provided that IP Source Routing is supported, a malicious router can place on the network a self-addressed packet, requesting either loose or strict source routing, and specifying either a logically removed or non-existent router as a required hop. Setting a legitimate and reachable router as the hop immediately preceding the unreachable one specifies the target of the attack. If intermediate routers only check whether the next hop is reachable, when the packet arrives at the router under attack, the packet will be dropped and the attacked router will be identified as bad for failing WATCHERS' *conservation-of-flow* test.

3.4 Fallible Assumptions

WATCHERS makes certain assumptions that allow the preceding attacks to occur. This section exhibits some of those assumptions and makes suggestions (potentially expensive) to improve the situation. The risks related to not addressing the assumptions and the feasibility of the suggested solutions are discussed.

3.4.1 Spoofing and Packet Modification

Assumption: Spoofing and packet modification will not occur.

In order for WATCHERS to function correctly, routers must not be allowed to spoof Administrative messages (WATCHERS, network topology, *etc*.) or modify packets as in sections 3.3.1, 3.3.2, and 3.3.4. If a bad router changes a packet's destination address without detection, the WATCHERS packet counters will not reveal any misbehavior. Additionally, if network topology messages can be spoofed, variants on the attacks in section 3.3 could prove devastating.

Possible solution:

WATCHERS currently verifies the integrity of its own communications. This must also be done for network topology messages. OSPF claims all messages authenticated, but two of its supported authentication options are “null” and simple password checking, both inadequate to prevent spoofing.

Packet integrity checking has been discussed extensively elsewhere [HPT97, SiKe97] and is an element of IPv6 [Huit98]. However, these features are not ubiquitous throughout the current Internet. Hence WATCHERS must include this as a requirement. See section 5.2.4 for a discussion of IPv6 and its impact on these scenarios.

3.4.2 Consorting Routers

Assumption: Consorting routers’ misbehavior is detectable using *per-destination* counters.

As shown in section 3.3.2, *per-destination* counters are insufficient to detect all consorting router misbehavior.

Possible Solution:

Augment WATCHERS' counters to be *per-source and per-destination.* This would significantly increase WATCHERS logging requirements: Given R as the number of routers in the AS, X as the number of external nodes (which without loss of generality may be taken as 1; see section 3.4.4), and N as the number of neighbors, then a router must then maintain $2N[(R+X)^2+2(R+X)]+N$ counters. In detail, each router would maintain 2 *T* counters with each neighbor, *per-source*, *per-destination* ($2N(R+X)(R+X)$ counters); 2 *S* counters with each neighbor, *per-destination* ($2N(R+X)$ counters); 2 *D* counters with each neighbor, *per-source* ($2N(R+X)$counters); and 1 *M* counter for each neighbor (N counters).

Because each router must also temporarily store the *T*, *S*, and *D* counter snapshots for itself, its neighbors, and their neighbors, in order to calculate total storage requirements, the number of these maintained counters is multiplied by a factor corresponding to the number of routers within 2 hops ((R+1) if all routers in the AS qualify). It should be noted that since messages are flooded, every router will need temporary space to hold the counter snapshots from every other router that sent one during the round; however, if the originating router is more than two hops away, the receiving node is able to deallocate the storage space after re-flooding the message if necessary.

For realistic networks where $N \ll R$ for all routers, and $X \ll R$, the number of counters maintained or stored would be $\boldsymbol{O}(R^2)$. In the worst case, for fully connected networks, as N approaches R-1, the number of counters maintained or stored would become $\boldsymbol{O}(R^4+R^3X)$.

For any topology, the communication cost for exchanging counters would be likewise affected. Since each router floods its *T*, *S*, and *D* counters, and each flooded message traverses each link in the AS, these create a multiplicative factor of *RL*, where *L* is the number of links in the AS. Applying this factor to the number of *T*, *S*, and *D* counters, the total counter exchange bandwidth consumption per round would be proportional to $2RL[(R+X)^2+2(R+X)]$. Assuming the digital signature and message overhead add a constant or negligible cost, this would be $\boldsymbol{O}(R^3)$ in a realistic network where $N \ll R$ for all routers, and $X \ll R$; however this would become $\boldsymbol{O}(R^6+R^5X)$ in a full AS (where $L=R(R-1)/2$). Note that these costs assume all counters are transmitted *as is*; it may be possible to significantly decrease the bandwidth usage by sending only non-zero counters, or applying another message compression scheme.

3.4.3 Observable Routing Actions and Methods

Assumption: All possible routing actions and methods are observable and appropriately validated by the WATCHERS protocol.

The WATCHERS protocol must also account for misrouted, forged, and modified packets. Whatever the situation, a router cannot simply drop packets, lest it become suspected of illegitimately doing so.

Additionally, the WATCHERS specification [BCPM+98b] does not indicate how to account for broadcast and multicast packets when updating its counters. Although an easy answer would be to eliminate these from analysis, this would open the door to bad routers dropping such packets undetected.

Possible Solution #1:

For misrouted packets, the receiving node should simply forward the packet to the next hop according to its routing table. To avoid accidental misrouting resulting from inconsistent views of the topology, routers could be required to synchronize their routing tables, and hold them constant, while packets were being exchanged. This would be quite expensive however, since it implies that no packets could be exchanged while routers disagree on the topology.

In any scenario, diagnosis of a bad router might include the type of malicious behavior detected, thus giving system administrators additional insight. For example, if routers are being accused primarily of misrouting (as opposed to dropping packets or disagreeing with their neighbors' counters), the source of the problem may only be a malfunctioning link-state protocol subsystem.

Possible Solution #2:

Packet modification and forging are not among the behaviors WATCHERS was designed to detect. In order to comply with the conservation of flow principle, these packets must also be sent on. Such packets are detectable using authentication and integrity checking. This is an issue for the destination node.

Possible Solution #3:

Broadcast and multicast packets do not observe the conservation of flow principle, as a single packet may induce the creation of numerous packets. Nevertheless, broadcast packets can be accounted for if the network topology consists only of pair-wise connections. Consider each broadcast packet sent over a link as a message originating with the sender and destined for the receiver. As WATCHERS currently ignores such

packets, to ensure they reach their destination, such packets could be *flooded*, as is done with WATCHERS' messages.

If, however, more than two nodes share a common link, *e.g.*, multiple machines connected to a single Ethernet repeater, only the MAC layer is aware of the originating network address [Davi88]. Here, two equally inelegant options exist. Either operate all WATCHERS routers in promiscuous mode, enabling them to obtain the originator's identity, or ignore broadcast packets entirely. If the originator's identity cannot be verified, a malicious router may resend a broadcast message just received, making it appear as though the message was sent twice from the same router. It should be noted that the WATCHERS model requires that the topology consist exclusively of pair-wise connections. An extension to handle this discrepancy is discussed in section 3.4.7.

Some routers pass multicast packets through as a single packet, while others replicate them into multiple packets. How to handle these packets is left as an open question.

3.4.4 Routers with External Links

Assumption: Routers that have external links are not required to be good

An *external link* is one that connects a *border router* to an external node. Any *border router* can arbitrarily drop packets to and from the external system(s) to which it is connected. Assuming authentication and *per-source* counters are not employed, it can alternatively substitute internal packets with externally-originated ones with the same destination or vice-versa.

Possible Solution #1:

It may not be appropriate to view the set of all external routers as a single external node (as [BCPM+98a, BCPM+98b] suggest) because as not all the external nodes may be interconnected. Instead, model each set of interconnected external machines as a single node. Just as the *good path condition* provides a trustworthy path within the WATCHERS network, we should require that each node connected to an external network also be good. This would also eliminate the issue of next-to-last routers conspiring with malicious *border routers* to drop packets [BCPM+98b].

Possible Solution #2:

It may prove difficult to determine the correct source or destination when a packet originates from, or is destined for, an external node. According WATCHERS' specification, when a packet originates from or is destined for an external node, the corresponding source or destination router is set to be the *border router* that accepted or transmitted the packet. This complicates updating WATCHERS' counters, especially if the external destination node is connected to multiple WATCHERS participants.

For complex routing involving source routing, a dynamic network topology, or a customized routing policy, a single packet may enter and exit the AS multiple times, and may even return to the originating AS. Thus, a packet entering an AS is not necessarily destined for that AS. Likewise, if the route to a packet's destination changes while the packet is in transit, that packet may pass through the originating node. With such routing, it is important that the packet cause the originator's appropriate *T* counter to be incremented, and not an *S* counter.

In any case, a router must know the routing policies of each of its neighbors in order to correctly identify misrouted packets. Since source routing itself may facilitate attacks, and may not be widely supported anyhow (see section 4.3.2.1), it may be best for networks running WATCHERS not to accept source routed packets at all.

3.4.5 Static Topology

Assumption: Inconsistencies in nodes' views of the network topology will be short-lived and will have only minor affects

Two problems arise. If all changes in network topology are broadcast using an unreliable protocol, such a message may be lost. This makes it possible for topological inconsistencies to persist. Although protocols such as OSPF guarantee that topology messages will be received, they do not guarantee their *timeliness*. As in section 3.3.4, a prematurely aged LSA may arrive before legitimate copies, nullifying any effect the latter might have.

Secondly, a malicious, undetected bad router can announce false topology changes at will. OSPF enforces certain delays in the transmission and acceptance of LSAs, which may in turn result in prolonged inconsistent views of the network topology among routers in the AS. Such disagreement is at the heart of accidental and intentional misrouting. Since WATCHERS itself influences the topology (by bringing down a shared link between itself and an identified bad router), it too is contributing to the potential disagreement.

Possible solution #1:

Do not allow the addition of new links or routers to an existing network running WATCHERS. This inhibits the creation of *ghost routers*, thus restricting the number of

good routers that can be falsely convicted as bad. However, in a ring network with only three nodes, if one is corrupt, it can still convince the other two that they are bad (see section 3.3.4).

Possible solution #2:

Regardless of the effect on the routing protocol's efficiency, it is necessary for WATCHERS' sake to attempt to minimize the period in which routers in the AS share inconsistent views of the network topology. To this end, the routing protocol must be configured for minimal delays in transmission and processing of topology updates.

Possible solution #3:

Maintain copies of each unique state of every neighbor's routing table during a round. During diagnosis, if a packet was routed correctly according to at least one of these states, assume it has been routed correctly. Although this may result in some false-negatives, *i.e.*, a bad router whose routing table has changed may continue to route packets using its old table, it eliminates false-positives that may result from legitimate topology changes, *e.g.*, a downed link corresponding to a bad router diagnosis.

3.4.6 Dynamic Packet Fields

Assumption: Routers will manipulate the dynamic fields in Administrative messages only as intended.

The dynamic nature of IP's TTL and OSPF's Age and Sequence Number fields must be accounted for. Currently, any router can alter these fields undetected. A malicious router can set the TTL on transit or sourced packets in order to deny their intended recipient and/or attack an intermediate router, which would be forced to drop a packet once the TTL expired. Similarly, a bad router can repeatedly set an LSA's Age

field to MaxAge, resulting in a thrashing topology if the originating router "fights back" as OSPF intends.

Possible solution #1:

In order to ensure no router (or group of routers) inappropriately increment or decrement such fields, all routers must be aware of the specific value, if any, each router is allowed to add or subtract. While TTL is always decremented by 1 for each hop taken, OSPF's Age field may be increased by an arbitrary amount on a per-interface basis. Once these offsets are known, all routers can compute what value a received packet's TTL and/or Age field should have, based on the path the packet is *assumed* to have taken (if we were to rely on a route recorded in the IP header, it too must have been integrity-checked).

Possible solution #2:

An expensive alternative (or addition) might be to keep *per-source per-destination* counters for every TTL value, and perform a modified *conservation-of-flow* test. This test would take into account that TTL should be decremented by 1 at each hop, *e.g.*, when x transient packets with a specific source/destination pair, each with a TTL of t, where $t>2$, are sent through an intermediate router, neighboring routers should see a total of x packets with that same source/destination pair leave that router with a TTL of $t-1$. (This method may need significant modification to handle OSPF's Age field, since the offset values are arbitrary.)

One potential attack remaining would be for an intermediate router to swap the TTL of two packets with the same source/destination pair. This can only be effective in causing packets to be dropped if some packets with the same source/destination pair

differ in their initial TTL and the destination can be made farther away from the source than the smallest initial TTL used.

Possible solution #3:

Self-reporting of dropped packets may be an effective strategy to balance conservation of flow, but consideration must be given to malicious routers monopolizing this power undetected. A threshold strategy might be applied to account for the legitimately dropped packets, but any non-zero threshold would allow for some misbehavior. If malicious routers are aware of such thresholds (or their methods of calculation, if dynamic), it may be possible for them to illegitimately drop packets undetected.

Possible solution #4:

Perhaps the simplest and most effective solution with respect to IP's TTL field, would be for every router in the AS to *not* send any packet with a TTL insufficient to reach its destination. This would break certain applications, *e.g.*, *traceroute*, which rely on such features. If this functionality were required, WATCHERS could be directed to ignore ICMP Echo Request messages, but this risks not detecting some malicious behavior. Some proposals avoid these issues by modifying *traceroute* to not use Echo Requests with insufficient TTL [Malk93].

However, if sufficient TTL was a requirement, and a packet was then received with an insufficient TTL, it would indicate that (1) the sending router was malicious, (2) its routing table disagreed with the recipient, or (3) both. In order to avoid dropping packets, which would still constitute malicious behavior, consideration might be given to TTL-boosting, although careful analysis is necessary before applying this remedy, in

order to avoid the infinite packet life and potential network saturation TTL was designed to prevent.

3.4.7 Link Failures

Assumption: Diagnosis will be performed on all participating WATCHERS nodes

If a link goes down during a particular WATCHERS round, and the attached routers attempt to send traffic over it, one or both routers may conclude the other is bad if the *validation* or *conservation-of-flow* tests are performed. For OSPF, unless a lower-level protocol informs it that a link is inoperative, RouterDeadInterval seconds must pass without hearing Hello packets before OSPF declares that link as down [Moy98].

Alternatively, if the diagnosis tests ignore traffic sent over a downed link during a round, any discrepancy between the attached nodes' counters will not be discovered. In this case, if a bad router can intentionally down its links for a portion of a WATCHERS round, it may escape analysis.

Possible solution #1:

As a precaution, whenever a link does down, perform the diagnosis. This may have the unfortunate side effect of causing connected good routers to be labeled as bad, but it ensures that a bad router cannot continue to capitalize on downed links.

Possible Solution #2:

If it is desirable to place the blame for a failed link where it is due, a modification to the WATCHERS model is suggested: treat all links as intermediate nodes. If a link fails, it corresponds to this intermediate node being bad. One caveat is that these intermediate nodes would not participate in the WATCHERS protocol themselves. As such, some router misbehavior would result in an associated link being blamed instead of

the guilty router. Despite this, detected bad routers will still eventually be removed from the network if they continue to misbehave.

With this modification, all links conform exactly to the *perfect transmission condition* with respect to *all* messages, *i.e.*, all transmissions sent to a neighboring node arrive intact with no delay. This holds because we can associate any actual delay, modification, or loss of data with the intermediate node. Additionally, multi-link and multiple-interface connections are more accurately represented using an intermediate node for each interface.

3.4.8 Thresholds

Assumption: Realistic discrepancies can be resolved through setting appropriate threshold levels

Network congestion, unreliable transport, message latency, and bad routers can all contribute to discrepancies in the counters used in the WATCHERS analysis. While good networks *should* only experience minor problems, when an undetected bad router exists on the network, it can exploit the problems above to generate false positives as in section 3.4. By induction, we must either set the thresholds high enough to ignore such noise, thus missing actual problems, or we accept false positives, possibly overlooking misbehavior.

Possible solution #1:

At a minimum, use a reliable transport mechanism, *e.g.*, TCP, for all Administrative communication. In this case, further steps should be taken to ensure prompt message delivery.

Possible solution #2:

Guarantee that Administrative messages are not dropped. TCP/IP currently permits a saturated node to drop packets. Selectively drop lower-priority packets and guarantee bandwidth sufficient for maximal high-priority usage. This scheme may enable bad routers to squeeze out lower-priority packets by saturating the network with Administrative messages, lending to DoS attacks. Such behavior may be discoverable by intrusion or anomaly detection systems, *e.g.*, [NePo99, VLRS99, Cann98, PoVa98, JGSW+97]. Alternatively, resource reservation protocols, *e.g.*, [BZBH+97], may accommodate bandwidth requirements while avoiding these DoS issues.

Neither of these solutions solves the latency problem. Rather, each focuses on limiting the damage potential of bad routers with respect to network congestion and transport reliability.

3.4.9 Synchronicity

Assumption: Messages are not passed simultaneously, and routers have no associated delay in WATCHERS' proof of correctness

Even if WATCHERS' four required conditions hold, plus the two additional proof requirements (the *perfect transmission condition* and *neighbor agreement condition*), a good router may still incorrectly diagnose another good router as bad.

As an example, say router *A* has already taken a snapshot of its WATCHERS counters, and then sends a *request* message to neighboring router *B*. Simultaneously, *B* sends a regular packet to router *A*. Upon receipt of the *request* message, *B* takes a snapshot of its counters. Because *A*'s *request* message and *B*'s packet were exchanged

concurrently, at least one of *A* and *B*'s counters will disagree, forcing each to declare the other as bad.

Although the *perfect transmission condition* requires that there be no delay between sending and receiving a WATCHERS message, the routers themselves may still delay in placing messages on the network. If they do, when a WATCHERS round begins and counter snapshots are taken, a node may still have a transient packet waiting to be sent. This packet will appear to be missing in the *conservation-of-flow* test. Again, a good router may potentially be incorrectly labeled as bad.

Possible Solution:

If we must guarantee that good routers are never falsely diagnosed as bad, we must ensure either the thresholds are sufficiently high or that no transient packets remain in the AS while snapshots are taken of WATCHERS' counters. The latter can be accomplished by additional synchronization among the participating routers, however, because the network would need to effectively shut down for a period of time, depending on the frequency or timing of the WATCHERS rounds, this may result in unacceptable delays.

3.4.10 Protocol Participation

Assumption: The WATCHERS participants either fully comply the RRR sub-protocol, or they do not participate in it.

There can be numerous actions taken by a malicious router to disrupt WATCHERS' messaging protocol. A bad router may choose to distribute multiple, varying accounts of its own counters for the same round. It might also attempt to replay, alter, or spoof other routers' messages.

Additionally, it's unspecified what to do about routers who communicate, or claim to have communicated, with bad routers. One might expect that routers should distinguish between accidental communication with a newly discovered bad router, and purposeful communication after the router is known to be bad.

Possible Solution #1:

To allow routers supporting the message flooding mechanism to determine whether or not a message they have received should be re-flooded, they must know whether it is an old or new packet. If only a sequence number and some form of windowing are used, as described in [Brad97], it must be done in such a way that all old packets have been flushed from the network before the window cycles around again.

"Current" Internet standards [Post81] estimate a packet in the Internet has a *maximum lifetime* on the order of tens of seconds; this figure was used in the design of protocols such as TCP to calculate certain timeout values (2 minutes is defined to be a packet's Maximum Segment Lifetime in the Internet). Unfortunately, the speeds at which modern communication links can send packets allow all 2^{32} TCP sequence numbers to be consumed on the order of *seconds*.[4,5] Avoiding such issues with stale packets in WATCHERS requires prudent selection of round length (thereby determining the rate of sequence number consumption) and the size of any windowing used.

Note that because a malicious router may indefinitely store a message it intends to replay, cryptographic keys used for authentication would have to be changed with each new window if a nonce was not employed. Use of a time-based nonce would require that

[4] [Post81] indicates that 2^{32} TCP sequence numbers may be consumed in 5.4 minutes at 100 megabits/sec. Assuming this rate of consumption scales linearly with bandwidth, a single OC-192 channel capable of 10 gigabits/sec could consume these same 2^{32} sequence numbers in only 3.24 seconds.

[5] This implies that each counter would most likely require more than 32 bits for storage when WATCHERS is run on a router with such high-bandwidth links.

routers' clocks were properly synchronized. Provided key generation is done frequently enough, it would also decrease threats related to compromised keys.

Possible Solution #2:

Note that it would be insufficient for a router to simply authenticate the WATCHERS messages destined for it, since a malicious router could then take advantage of the flooding mechanism: If a bad router broadcasted two different counter snapshots during the same round, and only their sequence numbers were used as the basis on which intermediate routers chose to re-flood them, the misbehavior would not be discovered. Such snapshots could be fashioned so as to manipulate other routers and their *conservation-of-flow* tests, potentially forcing incorrect diagnoses. Hence, every router must authenticate *all* WATCHERS messages it receives.

Possible Solution #3:

Should a router receive more than one unique, but authenticated request or response for a given round, originated by the same router, it can immediately identify the sending router(s) (which may include conspiring routers other than the source) as bad.

Possible Solution #4:

In order to handle misbehavior in the WATCHERS protocol itself, it is necessary that routers pay attention to extra details that serve to further cloud WATCHERS original purpose: to discover and isolate malicious routers that drop or misroute packets. While it is correct to require complete compliance with the protocol, unless a specification-based detection system is employed, *e.g.*, [BBK99, KRL97], it will be necessary to explicitly enumerate every conceivable form of misbehavior.

3.4.11 Computational Power

Assumption: The systems on which WATCHERS runs have enough computational power to keep up with the packet rate.

As with any time-sensitive application, WATCHERS must guarantee that it can timely process not only the packets for which it must update its counters, but also the Administrative messages it receives from other WATCHERS daemons.

Possible Solution:

At a minimum, each WATCHERS daemon must be capable of processing packets as quickly as the router that it monitors, taking into account that WATCHERS messages will require additional computation (each Administrative message must be authenticated, see section 3.4.10); it is expected that the bulk of effort expended processing each WATCHERS message will be spent performing cryptographic authentication.

Although [BCPM+98b] demonstrates that the cost of message processing (excluding authentication) is $\boldsymbol{O}(R^4)$ in a fully connected AS and $\boldsymbol{O}(R^2)$ in a sparse AS (respectively becoming $\boldsymbol{O}(R^5)$ and $\boldsymbol{O}(R^3)$ using *per-source per-destination* counters), processing normal packets may prove to be the greatest cost. In fact, WATCHERS would need to be capable of processing over *42 billion TCP packets per OC-192 interface per second*.[6]

Also, considering that *every* Administrative message must be authenticated, it may be possible for a malicious router to launch a DoS attack by overwhelming a router with such messages, before any action can be taken in response.

---

[6] Assuming maximal packet rate scales linearly with bandwidth, a 32-channel OC-192 with 10 gigabits/second per channel can transmit 2^{32} TCP packets in only 0.10125 seconds (see footnote 4 on page 39). Dividing 2^{32} by 0.10125 yields 42,419,430,080 packets/second.

3.5 Summary and Analysis

The original WATCHERS specification is elegant in its simplicity. However, the domain it intends to model is inherently complex and unreliable. This disparity reveals itself when attacks such as those described in this chapter are applied to WATCHERS.

Even when WATCHERS is not under attack, conservation of flow inherently fails for an unreliable protocol like IP. The underlying problem is that the conservation of flow equations do not take discarded packets into account. In particular, if a router drops a packet, it is assumed to be malicious. But IP packets may be discarded for a variety of reasons, many of which are behaviorally correct (such as the TTL expiring).

Assuming conservation of flow equations can be made to account for all messaging behaviors, a number of prerequisites must still be met. Even when WATCHERS' four required *conditions* hold, numerous problems remain. Among the more costly are the needs for integrity and authentication (digital signatures), a static network topology, and guaranteed, timely receipt of Administrative messages. Each of these alone would adversely affect the efficiency of any WATCHERS implementation or the network on which it is deployed.

Even without these expensive additions, questions remain as to how much WATCHERS' counters might disagree when *not* under attack. Specifically, the likelihood of in-transit packets appearing as dropped during the diagnosis phase is unknown. The ability to process packets quickly enough is also questionable, given the data rates possible with today's networking technologies. Some of these questions are addressed in chapter 4.

4 A WATCHERS Implementation

In order to better explore how realistic the attack scenarios and assumptions are in chapter 3, an implementation of the WATCHERS protocol was created. This chapter describes the implementation and some results of preliminary tests in which it has been used. The WATCHERS implementation and its source code, documentation, experimental test data, and list of known bugs are available [Hugh00].

4.1 Design Objectives

The major factors that influenced the WATCHERS implementation design were the testing objectives and the available equipment. As a primary testing objective, two major classes of experiments were desirable: attack and non-attack (benign) scenarios. As some of the attack scenarios discussed in section 3.3 would have required extensive modification to the routing protocol and possibly the supporting operating system as well; such tests were left for future work. Instead, the implementation was intended to be a proof-of-concept only, and was primarily subjected to the readily available benign test conditions during the preparation of this thesis. Section 4.2 reports the current status of the WATCHERS implementation.

The equipment available consisted of six x86 PCs and 100MB Ethernet hardware. The free distribution of RedHat Linux 6.2 was selected for its available modules: *gated* (supporting such routing protocols as RIP and OSPF), *tcpdump* (allowing *root*-user access to all network packets), and common utilities including *traceroute*, *ping*, *telnet*, and *ftp*. See chapter 8 for additional details on the network configuration.

4.2 Current Implementation Status

The core features of WATCHERS have been implemented. Close to 6,000 lines of C code in eight modules are compiled into the WATCHERS executable. This executable is run as *root* (*tcpdump* subprocesses require *root* privilege) on each of the participating systems. After parsing a configuration file, each WATCHERS process waits until it has established a TCP connection to each of its neighbors, and then pauses an additional 10 seconds. The protocol then commences with *request* messages to start the first round of WATCHERS. Configuration information and status messages are displayed by the user interface shown in Figure 4.1. All status messages are recorded to a log file for post-processing. Additional implementation details can be found in chapter 7.

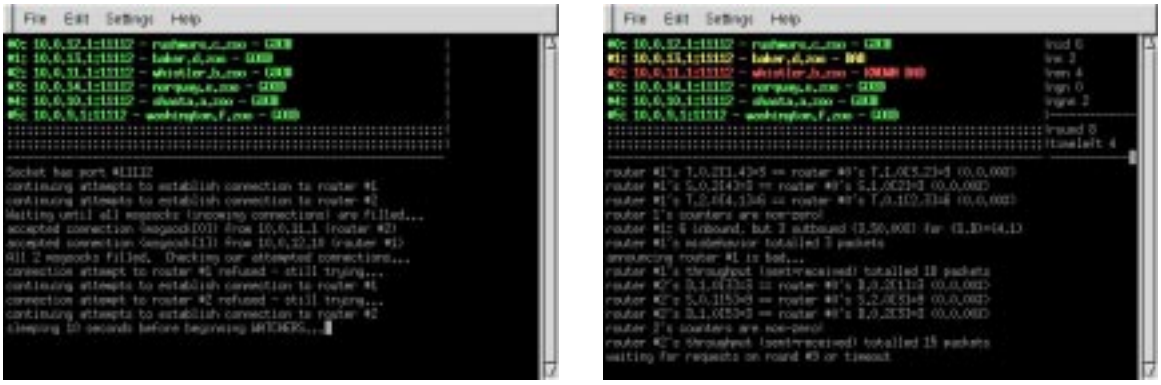

Figure 4.1: WATCHERS Implementation user interface screenshots.

4.3 Experimental Results

Each experiment is presented in four parts: its objectives, methods, results, and a brief interpretation. A more detailed summary and analysis is given in section 4.4. Unless noted otherwise, the following conditions applied to every experiment:

- Rounds were set to be 10 seconds in duration.

- All WATCHERS daemons were instructed to participate in the WATCHERS protocol and make accurate diagnoses. Although the term "daemon" is used, WATCHERS was run as a normal process.
- Misbehavior thresholds were set to zero (one dropped or missing packet indicated a bad router), however, for evaluation purposes only, daemons were instructed to continue to interact with any discovered bad routers.
- Only the *validation* and *conservation-of-flow* tests were performed (no attempt was made to identify misrouted packets).
- The network topology was that of Figure 4.2 and was assumed to be static; RIP was run in the background only to ensure each link remained operational.
- The only network traffic, besides that generated in the individual experiments, were RIP's 30-second periodic messages (which had little effect on WATCHERS performance, and no direct effect on its counters since broadcast messages were ignored).

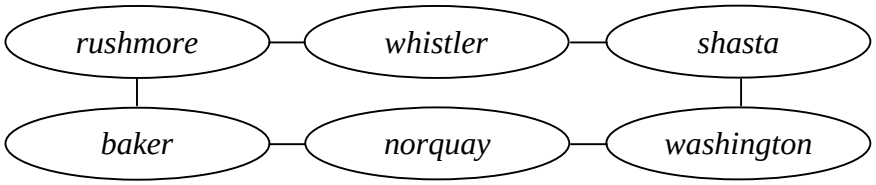

Figure 4.2: Ring-network topology used in WATCHERS implementation experiments.

Throughout the following sections, specific calculations and notation worth mentioning include:

- Percent Counter Disagreement = 100 * ($|r_1.C - r_2.C|$) / max($r_1.C$, $r_2.C$); the difference between the two values for the same counter C maintained by neighboring routers r_1 and r_2, divided by the larger of the two values and multiplied by 100.
- Percent Flow Unaccounted = 100 * abs(inbound – outbound) / (inbound + outbound); the difference between the inbound and outbound flow (calculated by summing the corresponding self-reported counter values), divided by the sum of the two flows, and multiplied by 100. Note that a non-zero Percent Flow Unaccounted could represent either a packet deficit or surplus.
- The notations $T_{r\_from,r\_to}[r\_source,r\_destination]$, $S_{r\_from,r\_to}[r\_destination]$, and $D_{r\_from,r\_to}[r\_source]$ are used to denote the *per-source per-destination* versions of the *T*, *S*, and *D* counters described in section 3.1.
- The notation *router<flow-source,flow-destination>* is used to denote the specific source and destination for a flow belonging to *router*.

4.3.1 Benign Conditions

The following tests were all performed on the ring network of good routers shown in Figure 4.2. WATCHERS *should not have* identified any malicious routers during these tests, as no packets were dropped. Most of the experiments in this section are presented in the order performed; the results of one experiment tended to influence the objective and conditions of the next.

4.3.1.1 Verification of WATCHERS Functionality

Objective: Verify the WATCHERS implementation's behavior and diagnoses are correct.

Method: By subjecting the network to "normal" traffic, *e.g.*, that generated separately by *ping*, *telnet*, and *ftp*, and observing WATCHERS does not identify the good routers as bad, this test will serve as a partial confirmation of the implementation's correctness.

Results: In the course of generating *ping* traffic (ICMP Echo Requests and Replies), it was discovered that in light traffic, the occasional packet appeared to have been dropped by an intermediate router in one round, but would then reappear in the next round (as a surplus packet) and continue to its destination. This occurred almost exclusively when an purposeful attempt was made to synchronize the traffic to coincide with WATCHERS' message exchanges. In heavy traffic, as caused by a file transfer or a large recursive directory listing, extreme counter disagreement rates (exceeding 50% of the larger value) were common. Failures of the *conservation-of-flow* test were also prevalent and extreme. In some cases, WATCHERS reported its router having sent more transient packets than it had received, during the first round in which it received any packets!

Interpretation: In light traffic, it became apparent that WATCHERS was observing all packets, but was identifying transient packets as dropped when they had not yet reached their destination. In the rounds following these identifications, the missing packets would appear to be injected by the same router, and again WATCHERS would report the resulting counter disagreement. This behavior was more readily induced when compiling WATCHERS without optimization flags, or by running additional CPU-intensive processes.

In light traffic, routers occasionally reported temporary packet deficits and surpluses as predicted in section 3.4.9. In some cases, these occurrences coincided with and could fully account for counter disagreement among neighbors. In heavy traffic however, the counter discrepancies could not be balanced in this way.

In the situations where more outbound than inbound transient packets were reported during the first round with packets, this seemed to indicate that WATCHERS was obtaining packets out of order. While the implementation did process packets in a round-robin fashion, not guaranteeing in-order processing, this cannot be the only factor, as the missing packet phenomenon was not symmetric with respect to Ethernet interfaces: The WATCHERS implementation checked for packets by polling the *tcpdump* process monitoring the interface to the left, and then the one to the right. If packets were missed due only to this ordering, counter values would have reflected this. Instead, in certain trials, the source and destination routers *both* claimed fewer *S* and *D* packets than their peers during the first round in which traffic was generated. Since the intermediate routers reported more traffic than the endpoints, yet only the endpoints generated packets, it is

clear that the WATCHERS daemons running on the endpoint routers were not being informed of all the packets which their router was sending and receiving.

Finally, discrepant counters did not balance out over successive rounds as they did in light traffic ("temporarily dropped" packets appeared as a deficit of one packet followed by a surplus of one packet in the next round), even after the packet generation had ceased. Although this might suggest routers were actually dropping packets, another explanation is due: Not all the packets being processed by the system were being reported *to* WATCHERS, in this case, by *tcpdump*. Evidence of this revealed itself when using *ping* in flood mode (using the *–f* command line switch), as *ping* reported no losses in receiving Echo Replies, while WATCHERS continued to indicate the same discrepancies.

4.3.1.2 Complications with Packet Timestamps

Objective: Determine whether packets are being received in order.

Method: By running *tcpdump* with its "unformatted timestamp" feature, out-of-order packets should be quickly identified, if they exist. A large file transfer using *ftp* was used to generate traffic from the same machine running *tcpdump*.

Results: Over 37,000 packets were reported in a 4.64-second time slice during the file transfer. Even when observing just the one Ethernet interface over which the traffic was sent, slightly more than 2% of all packets reported had a *negative* unformatted timestamp (*tcpdump* reported these timestamps as the elapsed time between when the kernel first began processing the current and previous packets.) The most extreme timestamps in this trial were -18 ms and 21 ms, while the average and standard deviation of the data set were 124 μs and 760 μs, respectively.

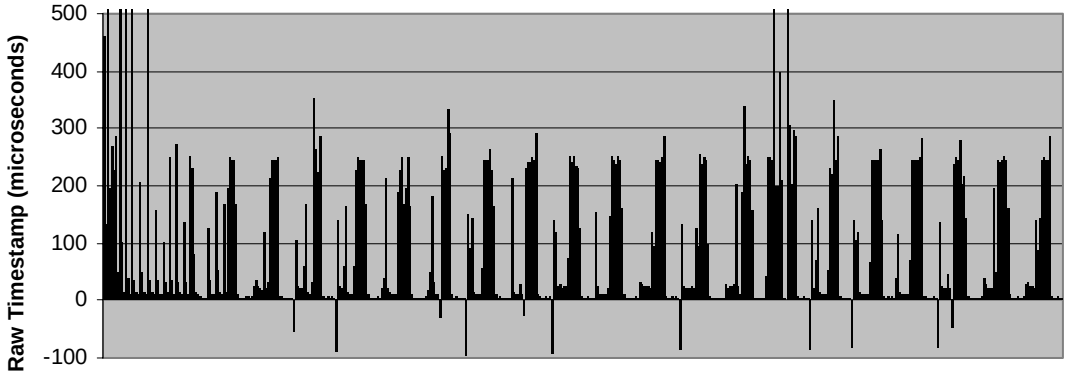

Figure 4.3: Raw timestamps reported by *tcpdump* for first 500 packets during file transfer. Note that not all packets' timestamps may be visible due to the printed resolution and that some positive values far exceed the scale shown.

Interpretation: Further investigation is needed to determine exactly why *tcpdump* would report negative timestamps at all when monitoring just one interface. If WATCHERS was forced to accept packets potentially out-of-order, an additional delay would be necessary when taking counter snapshots in order to ensure all packets corresponding to the previous round had been accounted for.

4.3.1.3 Unreported Packets as a Function of Process Priority

Objective: Determine whether the rate at which *tcpdump* drops packets can be improved by increasing process priority.

Methods:

(1) WATCHERS and its spawned processes (*tcpdump*s) were set to various priority levels between 0 (normal priority) and –20 (highest priority) by running WATCHERS using the command *nice*. *Ping* was set to generate a single burst of 2500 echo requests (using the *–f* and *–c* flags) from *whistler* to *norquay* during round #3;

(2) Same as (1), but only the *tcpdump* processes WATCHERS spawns were set to run at various priority levels, leaving the parent process running at its normal priority; and

(3) To verify that *tcpdump* itself would miss packets given minimal competition for CPU time, it was run separately without WATCHERS, using the command line "*tcpdump -l -n -q -t -x -s 20 -i eth0 icmp > /dev/null*". These parameters closely resembled those used by the WATCHERS implementation when it spawned *tcpdump* processes, except that here, only ICMP packets were filtered and standard output was redirected to /dev/null. *Ping* was again used to send bursts from *whistler* to *norquay*; for these trials, the burst size was varied between 10 thousand and 10.24 million Echo Requests (each with *ping*'s default 56-byte payload).

Results:

(In the figures that follow, the heavy black lines represent a linear interpolation of the unweighted average values at each priority level.)

(1) As seen in Figures 4.4 and 4.5, increasing process priority did have a positive effect on counter disagreement and unaccounted flow, but fell far short of eliminating these problems.

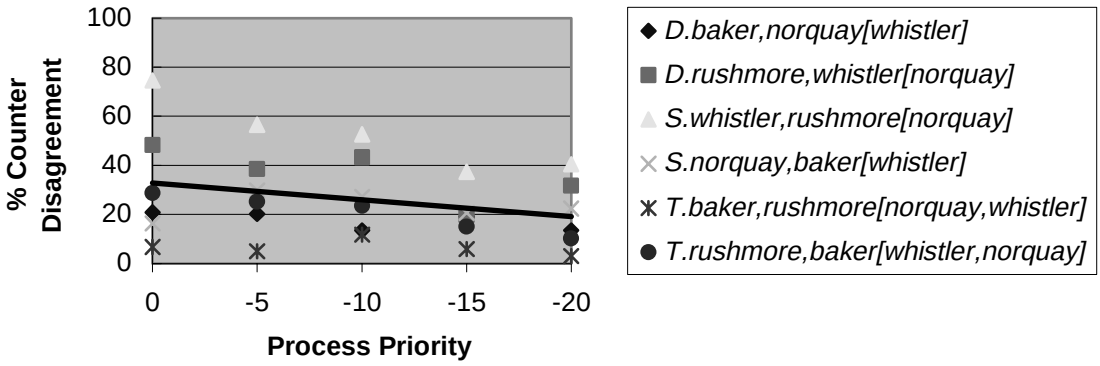

Figure 4.4: Percent Counter Disagreement as a function of WATCHERS' process priority.

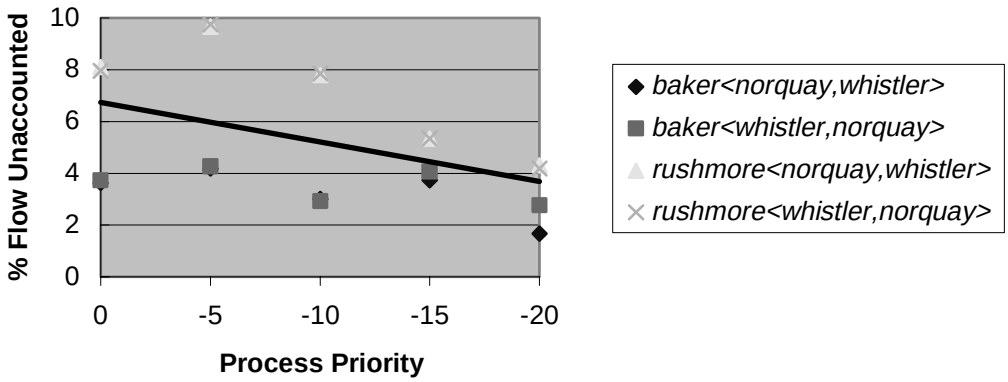

Figure 4.5: Percent Flow Unaccounted as a function of WATCHERS' process priority.

(2) For the same process priorities used in (1), Figures 4.6 and 4.7 show that improving the process priority of just WATCHERS' *tcpdump* subprocesses had an insignificant effect on the average counter disagreement and unaccounted flow.

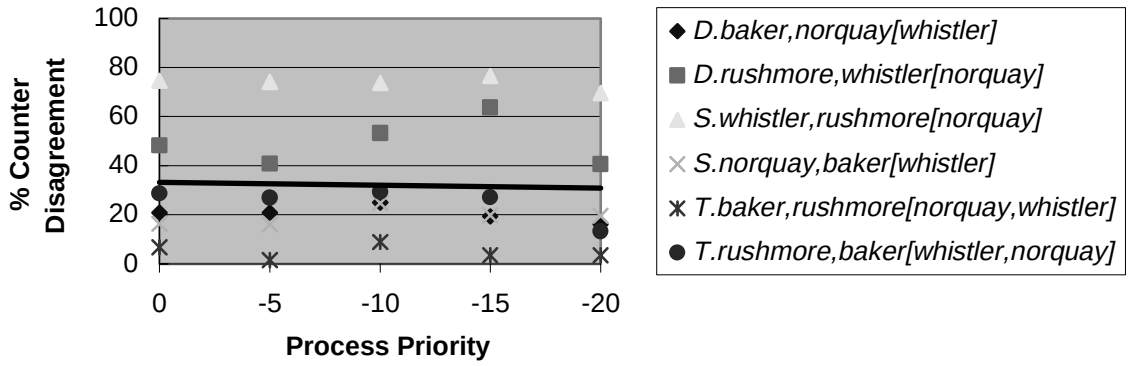

Figure 4.6: Percent Counter Disagreement as a function of *tcpdump*'s process priority.

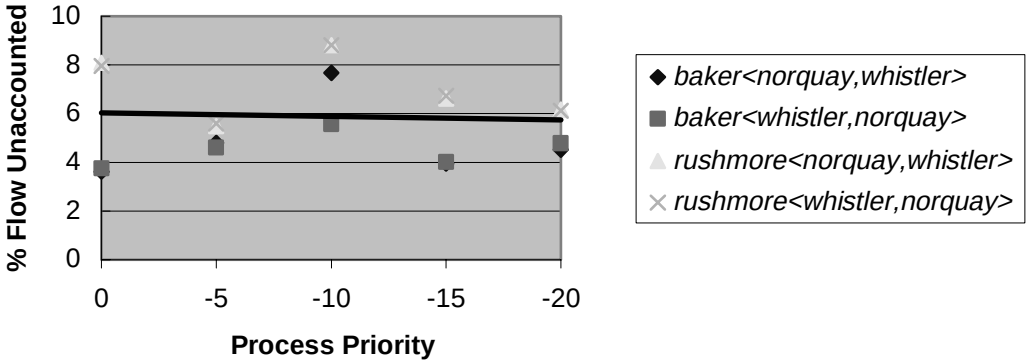

Figure 4.7: Percent Flow Unaccounted as a function of *tcpdump*'s process priority.

(3) As seen in Table 4.1, when *tcpdump* was run by itself, and its standard output redirected to /dev/null, it was able to filter significantly more packets than when spawned by the WATCHERS implementation. Here, maximizing process priority prevented *tcpdump* from missing packets except during high packet volume.

	Priority 0 (normal)		Priority –10		Priority –20 (highest)	
Echo Requests	Packets unreported by *tcpdump*	min/**avg**/max *ping* response times (ms)	Packets unreported by *tcpdump*	min/**avg**/max *ping* response times (ms)	Packets unreported by *tcpdump*	min/**avg**/max *ping* response times (ms)
10000	0	0.3/**0.3**/10.3	0	0.2/**0.3**/10.4	0	0.2/**0.3**/10.4
20000	0	0.3/**0.3**/10.3	0	0.2/**0.3**/10.4	0	0.2/**0.3**/10.3
40000	0	0.2/**0.2**/10.3	0	0.2/**0.2**/10.3	0	0.2/**0.3**/10.4
80000	0	0.2/**0.2**/10.5	0	0.2/**0.2**/10.3	0	0.2/**0.3**/10.3
160000	0	0.2/**0.2**/10.4	0	0.2/**0.2**/10.3	0	0.2/**0.3**/10.7
320000	0	0.2/**0.2**/10.8	*16004	0.2/**0.3**/20.2	*0	0.2/**0.4**/10.5
640000	0	0.2/**0.2**/11.3	*576	0.2/**0.2**/20.3	*0	0.2/**0.4**/10.5
1280000	513699	0.2/**0.3**/20.3	*34889	0.2/**0.3**/20.5	0	0.2/**0.3**/18.6
2560000	*257265	0.2/**0.2**/20.5	*132897	0.2/**0.4**/23.4	*23	0.2/**0.4**/20.1
5120000	**4508482	0.2/**0.4**/25.0	*217269	0.2/**0.3**/23.5	*0	0.2/**0.4**/19.7
10240000	*5542177	0.2/**0.3**/40.3	*431492	0.2/**0.4**/23.7	**3276	0.2/**0.5**/33.5

Table 4.1: Packets unreported by *tcpdump*, plus *ping* response times, at varying process priorities. Each asterisk indicates that *ping* reported having sent one more Echo Request than it had been instructed; these have been taken into account when calculating the packets unreported by *tcpdump*.

Interpretation: Methods (1) and (3) show that improving process priority can have a positive effect. This is worth studying to determine the optimal balance between parent and child priorities. However, method (3) also demonstrated that under high traffic volume, packets *would* eventually be lost, even when *tcpdump* is given the highest possible priority.

Although a real-time scheduling algorithm might improve drop rates, software will always be inherently slower than the available hardware on which it runs. It is therefore reasonable to expect that any WATCHERS implementation running on a modern PC and OS, *will not* be able to process all packets received by high-performance networking interfaces.

4.3.1.4 Unreported Packet Threshold as a Function of Burst Size

Objective: Determine whether there exists a packet rate threshold under which *tcpdump* reports all packets, and whether counter disagreement is bounded with increasing burst size.

Methods: Since increasing WATCHERS and its child processes' priorities did not eliminate counter disagreement, it is desirable to know at exactly what thresholds disagreement begins, and where it levels off. A single burst with a varied number of 56-byte payload Echo Requests was sent from *whistler* to *norquay* during each round.

Results: As seen in Figure 4.8, disagreement began between 410 and 415 Echo Requests per burst, and slowly leveled off above 10,000. Note that all counter disagreement leveled off below 60% except for packets originating with *whistler*. This may be due to the extra burden it carried as the originator of the Echo Requests.

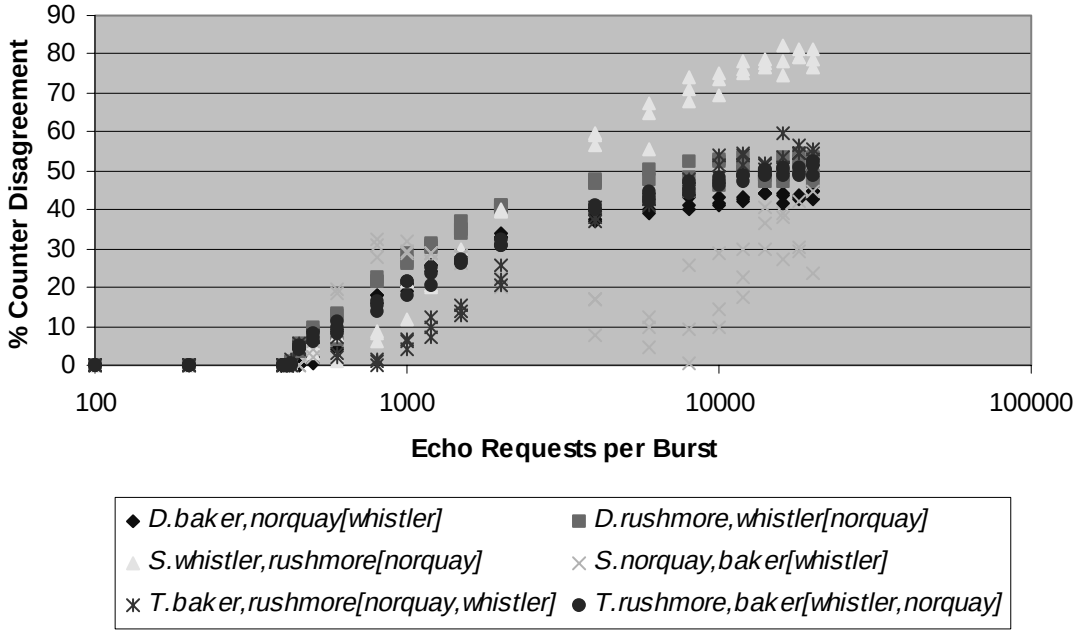

Figure 4.8: Percent Counter Disagreement as a function of Burst Size with one burst per round.

Interpretation: There is a clear threshold under which all packets are reported, however, its low value is quite unacceptable for high-speed networks. Likewise, the Percent Counter Disagreement levels off at extraordinarily high values. To make matters worse, the specific packet or burst rate at which this threshold occurs may be adversely affected by other factors such as CPU usage, traffic characteristics, *etc*.

4.3.1.5 Unreported Packets as a Function of Throughput

Objective: Determine whether there exists a correlation between unreported packets and the traffic rate.

Method: For this test, method (3) of section 4.3.1.3 was modified by varying the payload size of the Echo Requests while maintaining normal process priority.

Results: As shown in Table 4.2, increasing Echo Request payload actually decreased the number of dropped packets in general.

	54-byte payload		**512-byte payload**		**1024-byte payload**	
Echo Requests	Packets unreported by *tcpdump*	min/**avg**/max *ping* response times (ms)	Packets unreported by *tcpdump*	min/**avg**/max *ping* response times (ms)	Packets unreported by *tcpdump*	min/**avg**/max *ping* response times (ms)
20000	0	0.3/**0.3**/10.3	0	0.3/**0.3**/10.3	*0	0.3/**0.3**/10.3
40000	0	0.3/**0.3**/10.3	0	0.3/**0.3**/10.3	0	0.3/**0.3**/10.3
80000	0	0.2/**0.2**/10.3	0	0.3/**0.3**/10.3	0	0.3/**0.3**/10.3
160000	0	0.2/**0.2**/10.5	0	0.3/**0.3**/10.3	**0	0.3/**0.3**/10.3
320000	0	0.2/**0.2**/10.4	0	0.3/**0.3**/10.3	0	0.3/**0.3**/10.3
640000	0	0.2/**0.2**/10.8	***697	0.3/**0.3**/10.3	0	0.3/**0.3**/10.3
1280000	0	0.2/**0.2**/11.3	*0	0.3/**0.3**/10.3	**10	0.3/**0.3**/10.3
2560000	513699	0.2/**0.3**/20.3	*0	0.3/**0.3**/10.3	**72	0.3/**0.3**/10.3
5120000	*257265	0.2/**0.2**/20.5	**221572	0.3/**0.3**/10.3	***99836	0.3/**0.3**/10.3
10240000	**4508482	0.2/**0.4**/25.0	**160491	0.3/**0.3**/10.3	***176901	0.3/**0.3**/10.3
20480000	*5542177	0.2/**0.3**/40.3	***4291460	0.3/**0.3**/10.3	7x*2010203	0.3/**0.3**/10.3

Table 4.2: Packets unreported by *tcpdump*, plus *ping* response times, at varying Echo Request payload sizes. See note in Table 4.1's caption regarding asterisks.

Interpretation: The decrease in dropped packets with the increase in payload size seems to indicate that the kernel and/or *tcpdump* had more time to process packets with larger payloads. Since *tcpdump* only observed the first *n* bytes of each packet header, the longer packet payloads may have afforded more time for it to keep up with the bursts' packet rate, which appeared to saturate the Ethernet connection during all trials.

4.3.2 Attack Conditions

The following tests were performed with the expectation that WATCHERS daemons could be induced into declaring good routers as bad.

4.3.2.1 Source Routing

Objective: Verify the potential of the Source Routing attack in section 3.3.7.

Method: Simply by generating source routed packets as specified in section 3.3.7 and observing that they are dropped by the intended target would be sufficient evidence of the attack's viability. To generate source routed packets, *traceroute* was used with its "*–g*" flag to specify a gateway for loose source routing.

Results: Use of *traceroute*'s "*–g*" flag generated an error on the network of machines used in the previous experiments, and when used on the UC Davis campus network, it appeared as though loose source routed packets were not always routed as expected.

```
Trial 1 > traceroute 169.237.7.254
traceroute to 169.237.7.254 (169.237.7.254), 30 hops max, 40 byte packets
 1  169.237.7.254 (169.237.7.254)  0 ms  0 ms  0 ms

Trial 2 > traceroute 169.237.1.245
traceroute to 169.237.1.245 (169.237.1.245), 30 hops max, 40 byte packets
 1  169.237.7.254 (169.237.7.254)  1 ms  0 ms  0 ms
 2  169.237.246.238 (169.237.246.238)  1 ms  1 ms  1 ms
 3  area5-gw.ucdavis.edu (169.237.1.245)  3 ms  4 ms  10 ms

Trial 3 > traceroute -g 169.237.246.238 169.237.1.245
traceroute to 169.237.1.245 (169.237.1.245), 30 hops max, 52 byte packets
 1  * * *
 2  169.237.246.238 (169.237.246.238)  1 ms  1 ms  1 ms
 3  area5-gw.ucdavis.edu (169.237.1.245)  3 ms  4 ms  3 ms
 4  area5-gw.ucdavis.edu (169.237.1.245)  3 ms  3 ms  3 ms

Trial 4 > traceroute 198.32.249.53
traceroute to 198.32.249.53 (198.32.249.53), 30 hops max, 40 byte packets
 1  169.237.7.254 (169.237.7.254)  0 ms  0 ms  0 ms
 2  169.237.246.238 (169.237.246.238)  1 ms  1 ms  1 ms
 3  area5-gw.ucdavis.edu (169.237.1.245)  2 ms  2 ms  2 ms
 4  BERK--ucd5.ATM.calren2.net (198.32.249.53)  5 ms *  5 ms
/* 198.32.249.53 consistently dropped the 2nd packet sent to it */

Trial 5 > traceroute -g 169.237.246.238 198.32.249.53
traceroute to 198.32.249.53 (198.32.249.53), 30 hops max, 52 byte packets
 1  * * *
 2  169.237.246.238 (169.237.246.238)  1 ms  1 ms  1 ms
 3  area5-gw.ucdavis.edu (169.237.1.245)  8 ms  18 ms  11 ms
 4  BERK--ucd5.ATM.calren2.net (198.32.249.53)  14 ms *  6 ms

Trial 6 > traceroute -g 169.237.246.238 -g 169.237.1.245 198.32.249.53
traceroute to 198.32.249.53 (198.32.249.53), 30 hops max, 56 byte packets
 1  * * *
 2  169.237.246.238 (169.237.246.238)  1 ms  1 ms  1 ms
 3  area5-gw.ucdavis.edu (169.237.1.245)  3 ms  4 ms  3 ms
 4  BERK--ucd5.ATM.calren2.net (198.32.249.53)  6 ms *  6 ms

Trial 7 > traceroute 169.237.1.200
traceroute to 169.237.1.200 (169.237.1.200), 30 hops max, 40 byte packets
 1  169.237.7.254 (169.237.7.254)  0 ms  0 ms  0 ms
 2  169.237.246.238 (169.237.246.238)  1 ms  1 ms  1 ms
 3  * 169.237.246.238 (169.237.246.238)  2092 ms !H *

Trial 8 > traceroute -g 169.237.246.238 -g 169.237.1.200 198.32.249.53
traceroute to 198.32.249.53 (198.32.249.53), 30 hops max, 56 byte packets
 1  * * *
 2  169.237.246.238 (169.237.246.238)  1 ms  1 ms  1 ms
 3  * 169.237.246.238 (169.237.246.238)  2772 ms !H *

Trial 9 > traceroute -g 198.32.249.53 169.237.246.238
traceroute to 169.237.246.238 (169.237.246.238), 30 hops max, 52 byte packets
 1  * * *
 2  169.237.246.238 (169.237.246.238)  2 ms  2 ms  2 ms
 3  area5-gw.ucdavis.edu (169.237.1.245)  3 ms  3 ms  9 ms
 4  * BERK--ucd5.ATM.calren2.net (198.32.249.53)  5 ms  5 ms
 5  ucd1--BERK.ATM.calren2.net (198.32.249.30)  6 ms !S * *
```

Figure 4.9: Source routed packet trials using *traceroute*.

Regarding the trials in Figure 4.9, trials 1, 2, and 4 were normal traces with no source routing. Trials 3, 5, 6, 8, and 9 all used loose source routing, specifying at least one intermediate gateway as a required hop; it is interesting to note that in these trials, the

very first gateway in the route refused to return any ICMP message when it dropped source routed packets. Trials 7 and 8 attempted to trace to a non-existent machine, 169.237.1.200. In trial 8, this non-existent machine was specified as if it would be in the source routing attack of section 3.3.7; however, the result was the same as in trial 7 where no source routing was used. In trial 9, a loose source routing was specifically selected such that the packet would first pass through its destination before reaching the first required hop in its source route; this packet was not absorbed by its destination, despite having passed through it, and instead the next hop replied that the source routing had failed.

Interpretation: If source routing is supported, the attack of section 3.3.7 could be carried out. Transient packets might also be modified to be source routed, making it appear that another router was originating the attack. Unfortunately, in an AS only partially supporting source routing, dropped and misrouted source routed packets may pose a more complex problem, for even if a source routing attack was discovered, the inconsistent handling of source routed packets might assist malicious routers in avoiding identification. Source routing seems to create more problems for WATCHERS than can be tolerated, and unless justified by users' needs, source routing should probably not be supported.

4.3.2.2 Traceroute

Objective: Verify the potential of the Premature Aging attack of section 3.3.4.

Method: Although all routers in this experiment aged packets as they normally would, decrementing their value by one at each hop, the effect of the TTL running out in transit is the same as if the preceding router had maliciously altered it.

Results: As expected, each packet sent with a TTL insufficient to reach its destination was dropped, and an ICMP Time Exceeded message returned.

Interpretation: While this test relied on legitimately generated packets, a malicious router could easily modify the TTL of transiting packets and thus attack a router downstream by causing it to drop those packets. Preventative action must be taken to ensure a bad router cannot mount such an attack.

4.4 Summary and Analysis

The key finding in these experiments is that packets go unreported to WATCHERS when a sufficient amount of traffic is processed by the system, *even when the packets' destination application resides on the same system and does receive the packets*. Adjusting process priorities, compiler optimizations, and *tcpdump* settings does not adequately compensate, as *tcpdump* itself cannot keep up with high traffic volume. In order to obtain all packets, it will almost certainly be necessary to create a WATCHERS implementation *in hardware*. As claimed by one commercial router vendor, their ASIC is capable of filtering more than 20 million packets per second (pps), compared to software's 200 thousand pps capacity [Juni00].

Even when traffic rates are kept sufficiently low, allowing all packets to be reported to WATCHERS, there is still the issue of "temporarily dropped" packets. Although these may be influenced by implementation issues and delays in the message-passing mechanisms, the likelihood that there will be *some* transient packets still in transit when counter snapshots are taken, especially with heavy network traffic, seem to be quite good. With this consideration, the minimum misbehavior threshold necessary to eliminate these false-positives (falsely diagnosing a good router as bad) must be non-

zero, and is expected to be proportional to the number of packets a router could process in a single round. Further work is needed to verify this assertion.

To make matters worse, the attack scenarios displayed that source routing and premature aging attacks can be a significant threat. While source routing may not be widely supported, the apparent irregular handling of source routed packets may cause other dilemmas, either in WATCHERS' determination of the proper route for such packets, or for system administrators inundated with WATCHERS' warnings of related misbehavior.

5 Conclusion

WATCHERS has been shown to have some promise as a tool with which to detect malicious routers, however, there remain significant hurdles to overcome before it can be so applied. Namely, the quantity of false-positives exhibited is unsatisfactory, even on a network consisting only of well-behaved routers under modest load. If this issue is to be resolved, it will almost certainly necessitate running WATCHERS *in hardware*. This chapter distills the lessons learned and changes required to construct a viable WATCHERS implementation.

5.1 Modifications to the WATCHERS Algorithm

Based on the attack scenarios and related assumptions discussed in chapter 3, plus the empirical results in chapter 4 obtained with an actual WATCHERS implementation, it is clear that substantial changes to WATCHERS are required to strengthen its robustness in the presence of malicious routers. In this section, the significant results and suggested improvements are summarized, and updated pseudo-code of the diagnosis algorithm is provided.

While risk analysis must be applied when considering whether or not to adopt any of the suggested improvements provided herein, security tends to be an all-or-nothing approach, as attackers exploit the path of least resistance. Should any of the more serious weakness be left unaddressed, the resulting implementation would perform no better in the presence of a knowledgeable adversary.

5.1.1 Supplemental Required Conditions

Without additional requirements, the WATCHERS protocol can potentially fail irrespective of the number of malicious routers *and* the misbehavior thresholds. To address these issues, the original required *conditions* should be supplemented as follows:

5. *FIFO Condition*: All good routers process WATCHERS messages in a FIFO manner, such that the ordering of re-flooded messages is preserved.
6. *Reliable Transport Condition*: All Administrative messages are flooded via a loss-less acknowledgement-required mechanism.
7. *Source Routing Condition*: Source routed packets must not be supported, and should be neither sent nor accepted by each router.
8. *Border Router Good Condition*: Any router with at least one external link is required either to be good, or lie within another WATCHERS domain in which it is *insulated* by good routers, *i.e.*, there does not exist a path of bad routers from it to a *border router* of the same domain.

Figure 5.1: Supplemental WATCHERS required conditions.

5.1.2 Amelioration of Attacks and False-Positives

To prevent attacks and/or reduce false-positives, the following changes are recommended:

- Counters should be *per-source* and *per-destination*. (See section 3.4.2.)
- There should not be *simultaneous exchange*, as defined in section 3.3.5.
- WATCHERS should distinguish between a link going down, and a router forcing it down after discovering the neighbor is bad. Bad Router Announcement messages would solve this. (See section 5.1.4.)
- An upper bound should exist on the propagation delay of Administrative messages (see section 3.4.9). This would allow an operator to appropriately choose WATCHERS configuration parameters, *e.g.*, round length, misbehavior thresholds, *etc*.
- Disallow the addition of new links and new routers to the topology. This would prevent *ghost router* creation (see section 3.3.3). If new links and/or routers were then added, WATCHERS should be restarted.
- All messages must include a corresponding round number and nonce. Request messages would set the round number to the next round being agreed upon, response messages would indicate for which round the included counter snapshot applies, and Bad Router Announcement messages would indicate in which round

the originator identified the bad router. (See section 5.1.4.) Any time-based nonce would require synchronization among routers' clocks. (See section 3.4.10.)

- Each router should keep track of the bad routers it and its neighbors identify, in order to determine if misrouting and unauthorized communication have occurred. (See section 5.1.4.)
- Any ring segment must not have neighboring bad routers as nodes in that ring. (See section 3.3.5.)
- WATCHERS counters and round length must be chosen such that arithmetic overflow will not occur either by incrementing counters, or summations during diagnosis. (See section 3.4.10.) Along these lines, any Y2k-similar conditions must be addressed.
- Keys used to create digital signatures should be evolved periodically (see section 3.4.10).

5.1.3 Miscellaneous Improvements

The following suggestions may improve the overall quality of a WATCHERS implementation:

- Instead of requiring a majority of requests of all routers to start a new round, a majority need only be obtained of the non-isolated nodes (those not *logically removed* from the network).
- Counting packets instead of bytes may prove less expensive since less processing would be necessary on each packet header. MTU discovery [MoDe90, MJM96] and IPv6's disallowance of in-transit fragmentation eliminate the need to handle fragmented packets. (See section 5.2.4 for discussion of IPv6.)
- In addition to *supernodes* [BCPM+98b], scalability might be addressed by overlapping different WATCHERS domains – by at least two nodes, so that the *border routers* of each system are well contained inside the boundary of the other. In this way, these WATCHERS *chains* can extend the protocol's effective coverage while achieving bounded resource costs. (See section 5.2.2.)
- To permit programs such as *traceroute* to operate without adversely affecting WATCHERS, ICMP Echo Requests might be ignored. (See section 3.4.6.)

5.1.4 A Revised Diagnosis Algorithm

Taking into account the suggested improvements to WATCHERS, a revised pseudo-code diagnosis algorithm is presented in Figure 5.2. Noteworthy details are explained in line-by line comments following the presentation of the algorithm.

The revised pseudo-code for WATCHERS' diagnosis algorithm is as follows. Router r performs this diagnosis for the counter snapshots from round R. All counters are assumed to have non-negative values.

```
1  for each (n,*) ∈ r.CheckSet        /* r denotes the testing router, n the tested router */
2      AnnounceBadRouter(n);          /* n failed to identify its bad neighbor(s) */
3  r.CheckSet = Ø;
4  for each n ∈ {x| x ↔ r and ((x,*) ∉ r.BadSet)}
5      if (M_{n,r} ≠ 0) or (r has received at least 1 source routed packet from n)
6      then AnnounceBadRouter(n);
7      for each t∈ {x|x↔n and and ((x,*) ∉ n.BadSet)}
8          if (n and t exchanged at least 1 misrouted packet)
9          then r.CheckSet = r.CheckSet ∪ {(n,t)};
10     end
11 end
12 for each d,e∈Ð     /* Ð is defined as the set of all possible destinations */
13     for each n ∈ {x| x ↔ r and ((x,*) ∉ r.BadSet)}
14         /* check that n did not claim to have communicated with a bad router */
15         if (0≠∑_{∀t|t↔n and (((t,Q) ∈ n.BadSet)|Q<R-c)}        /* R is the round being analyzed */
16             (n.S_{t,n}[e] + n.S_{n,t}[e] + n.D_{t,n}[d] + n.D_{n,t}[d] + n.T_{t,n}[d,e] + n.T_{n,t}[d,e])
17         then AnnounceBadRouter(n);
18         /* local validation */
19         if (r.T_{r,n}[d,e] = n.T_{r,n}[d,e] ∧ r.S_{r,n}[e] = n.S_{r,n}[e] ∧ r.D_{r,n}[d] = n.D_{r,n}[d]) and
20             (r.T_{n,r}[d,e] = n.T_{n,r}[d,e] ∧ r.S_{n,r}[e] = n.S_{n,r}[e] ∧ r.D_{n,r}[d] = n.D_{n,r}[d])
21         then
22             if (∀t∈ {x|x↔n and ((x,*) ∉ n.BadSet)}        /* remote validation */
23                 (t's response message has been authenticated) and
24                 (t.T_{t,n}[d,e] = n.T_{t,n}[d,e] ∧ t.S_{t,n}[e] = n.S_{t,n}[e] ∧ t.D_{t,n}[d] = n.D_{t,n}[d]) and
25                 (t.T_{n,t}[d,e] = n.T_{n,t}[d,e] ∧ t.S_{n,t}[e] = n.S_{n,t}[e] ∧ t.D_{n,t}[d] = n.D_{n,t}[d]))
26             then    /* conservation-of-flow test */
27                 if ((n.S_{d,n}[e] + ∑_{∀t|t↔n and (((t,Q) ∉ n.BadSet)|Q<R-c)}(n.T_{t,n}[d,e])) ≠
28                     (n.D_{n,d}[d] + ∑_{∀t|t↔n and (((t,Q) ∉ n.BadSet)|Q<R-c)}(n.T_{n,t}[d,e]))
29                 then AnnounceBadRouter(n);
30             else for each t∈ {x|x↔n and ((x,*) ∉ n.BadSet)}
31                 if (t's response message has not been authenticated) or
32                     (t.T_{t,n}[d,e] ≠ n.T_{t,n}[d,e] ∨ t.S_{t,n}[e] ≠ n.S_{t,n}[e] ∨ t.D_{t,n}[d] ≠ n.D_{t,n}[d]) or
33                     (t.T_{n,t}[d,e] ≠ n.T_{n,t}[d,e] ∨ t.S_{n,t}[e] ≠ n.S_{n,t}[e] ∨ t.D_{n,t}[d] ≠ n.D_{n,t}[d]))
34                 then r.CheckSet = r.CheckSet ∪ {(n,t)};
35         else AnnounceBadRouter(n);
36     end
37 end
```

Figure 5.2: Diagnosis algorithm performed by router *r* using the counter snapshots from round *R*.

Line-by-line notes regarding the Diagnosis algorithm shown in Figure 5.2:

All: For simplicity of exposition, the Diagnosis algorithm shown assumes zero-tolerance, *i.e.*, all misbehavior thresholds are zero. This can be easily extended. Additionally, it should be assumed that any deviation from the WATCHERS protocol, *e.g.*, flooding the same message more than once, flooding messages with pre- or post-dated round numbers, *etc.*, should be considered misbehavior and flagged as such. Enumeration, implementation, and debugging of these special cases are left as exercises for the reader. ;) Seriously, specification-based monitoring may ease this burden and should be explored as a potential solution [BBK99, KRL97].

Line 1: *r*.CheckSet is updated as Bad Router Announcement messages arrive, removing any entry in *r*.CheckSet corresponding to the originator and bad router. The two elements in each ordered pair added to *r*.CheckSet correspond to the neighbor who should identify a bad router, and the router it should identify as bad. For clarity, *r*'s ownership of CheckSet is explicitly shown by prepending "*r*.", as is done with BadSet and counter values, despite the fact that *r* does not maintain a CheckSet for any router other than itself.

Line 2: AnnounceBadRouter(*n*) sets *r*.BadSet = *r*.BadSet $\cup$ {(*n*,*R*)} and floods a Bad Router Announcement indicating *n* is bad.

Line 4: An asterisk matches any element, and "$\leftrightarrow$" is defined as "neighbors". In contrast to [Brad97, BCPM$^+$98a, BCPM$^+$98b], here "$\leftrightarrow$" is a relationship based on physical, not virtual, connection, *i.e.*, if a shared link goes down (or is forced down), the two routers at its endpoints are *still neighbors* so long as the cable or transmission medium has not been permanently disconnected. This difference is duly compensated with the use of BadSet and Bad Router Announcements. It is important to note that without this change, as written, this line would excuse routers who misrouted packets, but whose shared link became inoperative during that round.

Line 8: This covers both routers depicted in the misrouting consorting router scenario presented in [BCPM$^+$98b]. If this diagnosis is to be done for rounds in which the topology changed, then each router must remember every unique state of the routing tables of each of their neighbors; then a packet would not be misrouted if it had been routed correctly according to at least one of the states of the corresponding routing table.

Line 12: $\eth$ includes not only the routers participating in the WATCHERS protocol, including bad routers intended to participate, but also any external nodes that are defined. (See section 3.4.4.) As line 12 is written, some work within the *for each* clause is duplicated, *e.g.*, comparing counter values by using only one of the two variables, *d* or *e*; these inefficiencies are intentional, as they provide more comprehensible pseudo-code, but should be rectified for any optimized implementation.

Line 15: If a neighbor reports having communicated with a router it learned was bad in the previous round, then it too is bad. The constant *c* is included for completeness, as the

appropriate value of *Q* depends on any propagation delay that might exist (depending on when a Bad Router Announcement is sent – this could be done in other stages of the protocol as packets arrive, *e.g.*, source routed packets – entries might be added to *n*.BadSet during a different round than n learned of the bad router). Note that *n*.BadSet is not modified during the diagnosis phase; it is intended to be updated during message exchange, when r receives a Bad Router Announcement, or acknowledgement of

Line 19: "$\wedge$" is the logical "and" operator.

Line 23: "*t's message has been authenticated*" means "one and only one unique authenticated message from *t* for round *R* was received." Authentication includes verification both of the sender's identity and the integrity of the message, as well as proper decoding according to the agreed upon message format. Note that while not explicitly checked, should these conditions cease to hold for rounds < *R*, *e.g.*, at least one authenticated message was received after diagnoses in which different counter values were claimed, this too should be considered misbehavior. Similar steps should be taken with respect to all WATCHERS messages.

Line 32: "$\vee$" is the logical "or" operator.

With the recommendations presented and the pseudo-code outlined above, we now have many reasons why a router may be identified as bad:

- A neighbor's failed the *validation* test.
- A neighbor failed the *conservation-of-flow* test.
- A neighbor misrouted a packet (or conspired to do so).
- A neighbor flooded a duplicate of a message it already flooded once before.
- A neighbor flooded a message that could not be authenticated.
- A neighbor's request or response message was never received and authenticated.
- A neighbor originated an authenticated, but *incomplete* message (or the neighbor re-flooded such a message it was required to decode itself).
- A neighbor flooded multiple versions of the same response message for a single round.
- A neighbor flooded a message *out of sequence* (either by disobeying the *FIFO condition*, or .by sending a packet with an inappropriate round number).
- A neighbor claimed to have communicated with a non-neighbor or a neighbor it had previously known to be bad.
- A neighbor failed to take down the shared link with at least one of its neighbors it should have identified as bad.
- The catchall: A neighbor did not comply with WATCHERS communication protocol. The above enumeration should by no means be considered complete, and a specification-based approach is recommended to identify additional misbehavior not described herein, *e.g.*, [BBK99, KRL97].

5.2 Future Work

While the WATCHERS implementation suffers from serious performance issues, and the protocol itself is wrought with problems related to its model, it would be premature to discount WATCHERS as useless. In this section, three areas of future work are identified: improvements to the protocol, improvements to the implementation, and further experiments. Additionally, consideration is given to the impact that IPv6 might have on the WATCHERS protocol.

5.2.1 Improvements to the WATCHERS Protocol

In order to be successful, it is necessary that the WATCHERS protocol make few false-positive diagnoses under benign conditions. Legitimately dropped packets must be recognized as such, and broadcast and multicast packets must be accounted for, as these are all behaviors typical of today's networks. It is clear that the concept of conservation of flow as defined herein does not hold in today's Internet; adaptations to the flow model, *e.g.*, [LMPS98], might be considered.

Even with an accurate model of network flow, latency and simultaneous exchange might still pose problems in benign networks, although their effect may not as significant as that created by desynchronized WATCHERS participants. Effective yet inexpensive solutions to these problems would constitute valuable future work.

Another invaluable contribution would be an efficient and correct method to determine the source and destination of each packet. As discussed in section 3.4.4, these may not correspond to the source and destination addresses listed in the packet header.

Perhaps the most serious issue facing WATCHERS is it's potential in assisting attackers. It has been asserted that WATCHERS does not open the routing infrastructure to new vulnerabilities [Brad97]. However, even if the attacks in section 3.3 were prevented, any temporary router or link failure might result in permanent disconnection from the network as a result of WATCHERS' diagnosis. The history of DoS vulnerabilities in routers makes this threat quite realistic [Cisc00]. What would otherwise be a brief disruption in service could be catastrophic if WATCHERS were employed. To avoid undue burden on network administrators, it is necessary that WATCHERS include a means by which "bad" routers might automatically rejoin the network.

Unfortunately, even if WATCHERS can be made to function well in the presence of only good routers, in order to accurately detect malicious routers, a number of more challenging issues must be overcome. Among these are the need for a message authentication scheme that facilitates intermediate-host authentication, and an integrity scheme allowing robust manipulation of OSPF's Age and IP's TTL fields (or at least detection of their misuse).

Some solutions to the aforementioned problems have been discussed in previous chapters, but many are quite inefficient, and some have not been explored in terms of their cost to the protocol. Again, effective yet efficient solutions would prove invaluable to WATCHERS.

If false-positives cannot be eliminated, the design and analysis of an algorithm to amortize these errors related to simultaneous exchange and in-transit packets, yet still identify misbehavior with a zero- or low-tolerance threshold would constitute valuable future work.

Finally, it would be worthwhile to investigate whether attack behavior can be identified by intrusion detection or specification-based systems. If so, WATCHERS need not concern itself by adopting potentially expensive remedies, and the protocol might retain some of its simplicity. In any case, adoption of WATCHERS as a *sensor* may provide otherwise unavailable data to correlation engines, *e.g.*, [SCCD+96, Goan99]. These possibilities should be researched further.

5.2.2 Improvements to the WATCHERS Implementation

Unreported packets are certainly the most important problem to address in the WATCHERS implementation. It may be possible to hook into the network stack as *tcpdump* does with *libpcap*, with the hope that more packets could be observed. Additionally, a real-time scheduling algorithm may be of assistance. Common problems with packet filters are discussed in [Paxs97a].

The best solution however, would be to implement WATCHERS in hardware. This would resolve the unreported packet issue both by providing immediate access to the packets, as well as ensuring that processing speeds would match or exceed those necessitated by high-bandwidth interfaces.

But before resources are expended on a hardware venture, additional improvements should be made to the existing software implementation. More rigorous testing is required and the known bugs should be fixed. Various opportunities for DoS attacks and non-compliance with the WATCHERS protocol by malicious routers should be precluded and detected.

Integration with the routing protocol is also required, and routing information must be maintained locally for each neighbor so that misrouting may be detected.

Additionally, Bad Router Announcements should accompany any positive diagnosis of malicious activity.

A necessary addition is the use of digital signatures for authentication. The most appropriate algorithm, key length, and key management scheme must be chosen. These would almost certainly depend on available hardware and such implementation parameters as round length and message size.

The supplemental *conditions* of Figure 5.1 must also be met. The *FIFO* and *reliable transport conditions* may compel the use of QoS mechanisms, as FIFO queuing with loss-less and bounded-delay packet delivery has been shown to be achievable using source rate control and appropriately sized buffers [CFZF98].

WATCHERS *Chains* might afford scalability without proportional expense in messaging, and would complement the *border router good condition*. *Supernodes* could help relax the *good neighbor condition* in the fine-grained sense. Both *chains* and *supernodes* might benefit from the proposed OSPF for IPv6 *Instance IDs* [CFM99]. Lastly, certain tasks such as counter updates, WATCHERS messaging, and diagnosis may be conducive to parallelization.

5.2.3 Additional Experiments

Additional tests should be run, including the yet untested attack scenarios from section 3.3. Future tests should also focus on gauging WATCHERS' performance and optimal thresholds.

5.2.3.1 Performance

In order to accurately gauge WATCHERS' performance, realistic background traffic must be used. Traffic composition will certainly vary widely depending on the environment, and the WATCHERS implementation should therefore be exposed to as many environments as possible. Of particular interest would be the number of non-zero counters, as this would influence opportunities for message compression, directly affecting bandwidth consumption during counter exchange. The network topology must be considered as a factor in performance [ZCD97]. Real production networks should be used, and if possible, WATCHERS should be configured to run on actual routers.

Actual CPU usage and bandwidth consumption should be measured for counter updates, key management, and authentication. The impact of varying parameters such as network topology, AS size, and round length should also be determined.

Finally, it may be useful to determine if the expense of an AS-wide packet flush would be too inhibitive. If not, these might be used to allow snapshots to be taken in the absence of in-transit packets.

5.2.3.2 Thresholds

With the current WATCHERS implementation, misbehavior thresholds would need to be embarrassingly high in order to avoid making false-positive diagnoses. Such high thresholds would easily allow malicious routers to "fly under the radar." Preliminary data suggests thresholds might need adjustment proportional to the network traffic, but perhaps this is only true when the link or node is near saturation. If the

number of unreported packets could be significantly reduced, this question could be more definitively answered.

It is likely that a non-zero threshold will be required. In this case, a variety of methods are available; appropriate thresholds may be fixed, dynamic, per-misbehavior-type, *etc*. Hybrid approaches such as *layered window* tests might also be explored: Keep *n* rounds' worth of data, and perform *conservation-of-flow* analysis on the sum of these rounds' counters. Dynamic or perhaps simply lower thresholds might be applied in this way to achieve fewer false-positives.

The holy grail of detection systems is zero-tolerance. All data gathered thus far seems to indicate that this goal will remain elusive for WATCHERS. Even still, it would be beneficial to determine a concrete upper bound for misbehavior thresholds that would yield no false-positive diagnoses. Consideration will need to be given to the frequency of non-malicious routing errors ([LMJ98, Paxs97b] provide empirical data on Internet routing pathology).

5.2.3.3 Other Parameters

As mentioned above, WATCHERS round length should be varied in many contexts. Discovering a realistic lower bound on round length may prove useful in understanding the costs and delays in WATCHERS messaging.

An additional correlation of interest might be the percentage of unreported packets as a function of round length (or frequency). Increasing the round length should reduce the percentage of unreported transit packets, but would also lead to a longer delay in bad router discovery. This tradeoff should be further explored.

Finally, the precise OSI level at which WATCHERS should operate has not been explored. A clear advantage of operating at the network level is that the source and destination IP addresses are available and uniquely identify the source and destination nodes. Operation above the network layer is unrealistic, as routers and switches do not implement these [Kesh97], but operation at the Datalink or Physical layers might be appropriate in limited applications. As illustrated in section 3.4.3, if more than two nodes share a common link, *e.g.*, an Ethernet repeater, determination of a packet's source address will require operation at a lower OSI level.

5.2.4 Implications of IPv6

Although IPv6 promises greater reliability and security, its use still leaves WATCHERS vulnerable to many of the attacks in chapter 3. Specifically, IPv6 can detect both header and payload modification, but only at the destination, and only when the Authentication Header and Encrypted Security Payload are used. The result is that modified packets may cause intermediate WATCHERS routers to incorrectly increment their counters. This is consistent with the "steel pipe" analogy: only the source and destination can be confident of message authenticity and integrity. It is suggested that intermediate source address verification may be accomplished by some variant of the Authentication Header, but this remains an open question [Huit98].

Introducing another potential attack, IPv6 routers will not fragment packets already on the network, but will instead drop them and return an ICMP message if they are too large for the next hop. Again, because WATCHERS does not consider this ICMP message as compensation for the dropped packet, the *conservation-of-flow* test will fail.

Another aspect of IPv6 is that its OSPF link-state database will not be shared with the IPv4 database; IPv6 OSPF and IPv4 will run in parallel, significantly increasing WATCHERS' memory and computational requirements on an AS supporting both IPv4 and IPv6.

Two offsetting simplifications IPv6 offers are a single 32-bit identifier for each router, independent of its network addresses, and its intolerance for in-transit fragmentation. Per the latter point, it may be sufficient for WATCHERS to simply count packets instead of bytes. Since malicious routers could always balance the byte count for packets they alter during a WATCHERS round, there would no longer be any advantage to counting bytes on a network using only IPv6, or one in which in-transit fragmentation was not permitted.

Note that while TCP/IP scenarios have been discussed exclusively, WATCHERS suffers similar shortcomings when applied to IPX/SPX, SNA, and other network protocols, due to their similarities to TCP/IP.

6 References

[BBK99] R. Buschles, M. Borning, D. Kesdogan, "Transaction-based Anomaly Detection," *Proceedings of 1999 Workshop on Intrusion Detection and Monitoring* (April 1999). pp. 129-140.

[BCPM+98a] K.A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, R.A. Olsson, "Detecting Disruptive Routers: A Distributed Network Monitoring Approach," *Proceedings of the 1998 IEEE Symposium on Security and Privacy* (May 1998). pp. 115-124.

[BCPM+98b] K.A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, R.A. Olsson, "Detecting Disruptive Routers: A Distributed Network Monitoring Approach," *IEEE Network* (September/October 1998). pp. 50-60.

[BeLa73] D.E. Bell, L.J. LaPadula, *Secure Computer Systems: Mathematical Foundations and Model*, M74-244, The MITRE Corp., Bedford, MA (May 1973).

[BMR97] N. Brownlee, C. Mills, G. Ruth, *Traffic Flow Measurement: Architecture*, RFC 2063 (January 1997).

[Brad97] K.A. Bradley, *Detecting Disruptive Routers: A Distributed Network Monitoring Approach*, Master's Thesis, University of California, Davis (September 1997).

[BZBH+97] R. Branden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, *Resource ReSerVation Protocol (RSVP)* -- Version 1 Functional Specification, RFC 2205 (September 1997).

[BZT99] R. Bettati, W. Zhao, D. Teodor, "Real-Time Intrusion Detection and Suppression in ATM Networks," *Proceedings of 1999 Workshop on Intrusion Detection and Monitoring* (April 1999). pp. 111-118.

[Cann98] J. Cannady, "Artifical Neural Networks for Misuse Detection," *Proceedings of the 21st National Information Systems Security Conference*, (October 1998). pp. 443-456.

[CFM99] R. Culton, D. Ferguson, J. Moy, *OSPF for IPv6*, RFC 2740 (December 1999).

[CFZF98] I. Chlamtac, A. Faragó, H. Zhang, A. Fumagalli, "A Deterministic Approach to the End-to-End Analysis of Packet Flows in Connection-Oriented Networks," *IEEE/ACM Transactions on Networking*, 6(4) (August 1998). pp. 422-431.

[Cheu97] S. Cheung, "An Efficient Message Authentication Scheme for Link State Routing," *Proceedings of the 13th Annual Computer Security Applications Conference* (December 1997). pp. 90-98.

[ChLe97] S. Cheung, K.N. Levitt, "Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection," *Proceedings of the 1997 New Security Paradigms Workshop* (September 1997). pp. 94-106.

[Cisc00] *Cisco Internet Security Advisories*, Cisco Systems, http://www.cisco.com/warp/public/707/advisory.html, (May 2000).

[Come95] D.E. Comer, *Internetworking with TCP/IP, 3rd Edition, Volume I*, Prentice Hall, Upper Saddle River, NJ (1995).

[CSI00] *2000 Computer Crime and Security Survey*, Computer Security Institute, http://www.gocsi.com (March 22, 2000).

[Davi88] J. Davidson, *An Introduction to TCP/IP*, Springer-Verlag, New York, NY (1988).

[FSPS95] C. Farrell, M. Schulze, S. Pleitner, M. Songerwala, "Network Monitoring and Visualization," *Internetworking: Research and Experience*, 6(4) (1995). pp. 167-184.

[Goan99] T. Goan, "A Cop on the Beat: Collecting and Appraising Intrusion Evidence," *Communications of the ACM*, 42(7) (July 1999). pp. 46-52.

[HAB00] J.R. Hughes, T. Aura, M. Bishop, "Using Conservation of Flow as a Security Mechanism in Network Protocols," *Proceedings of the 2000 IEEE Symposium on Security and Privacy* (May 2000). pp. 132-141.

[HPT97] R. Hauser, T. Przygienda, G. Tsudik, "Reducing the Cost of Security in Link-State Routing," In *Proceedings of the 1997 Symposium on Networked and Distributed System Security* (February 1997).

[Hugh00] J.R. Hughes, WATCHERS Implementation Web Site, http://seclab.cs.ucdavis.edu/~hughesj/watchers/ (June 2000).

[Huit98] C. Huitema, *IPv6: The New Internet Protocol*, Second Edition, Prentice-Hall, Inc., Upper Saddle River, NJ (1998).

[Huit00] C. Huitema, *Routing in the Internet, 2nd Edition*, Prentice Hall, Upper Saddle River, NJ (2000).

[JGSW+97] F. Jou, F. Gong, C. Sargor, S.F. Wu, R. Cleveland, *Architecture Design of a Scalable Intrusion Detection System for the Emerging Network Infrastructure*, Technical Report E296, Advanced Network Research, MCNC (April 1997).

[Juni00] *Internet Processor II Press Release*, Juniper Networks, http://www.juniper.net (April 17, 2000).

[Kesh97] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley, Reading, MA (1997).

[KRL97] C. Ko, M. Ruschitzka, K. Levitt, "Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach," *Proceedings of the 1997 IEEE Symposium on Security and Privacy* (May 1997). pp. 175-199.

[LMJ98] C. Labovitz, G.R. Malan, F. Jahanian, "Internet Routing Instability," *IEEE/ACM Transactions on Networking*, 6(5) (October 1998). pp. 515-528.

[LMPS98] A. Lombardo, G. Morabito, S. Palazzo, G. Schembra, "Flow Theory: an Enhancement," *Proceedings of the 1998 International Conference on Network Protocols* (October 1998). pp. 129-136.

[Malk93] G. Malkin, *Traceroute Using an IP Option*, RFC 1393 (January 1993).

[Malk98] G. Malkin, *RIP Version 2*, RFC 2453 (November 1998).

[Merc00] *Internet Siege Becomes a Boon to Companies that Sell Security*, newspaper article, San Jose Mercury News (February 11, 2000).

[MJM96] J. McCann, S. Deering, J. Mogul, *Path MTU Discovery for IP version 6*, RFC 1981 (August 1996).

[MoDe90] J.C. Mogul, S.E. Deering, *Path MTU Discovery*, RFC 1191 (November 1990).

[MoKa97] I.S. Moskowitz, M.H. Kang, "An Insecurity Flow Model," *Proceedings of the 1997 New Security Paradigms Workshop* (September 1997). pp. 61-74.

[Moy98] J. Moy, *OSPF Version 2*, RFC 2328 (April 1998).

[NePo99] P.G. Neumann, P.A. Porras, "Experience with EMERALD to Date," *Proceedings of 1999 Workshop on Intrusion Detection and Monitoring* (April 1999). pp. 73-80.

[Paxs97a] V. Paxson, “Automated Packet Trace Analysis of TCP Implementations,” *Proceedings of the ACM SIGCOMM* (1997). pp. 167-179.

[Paxs97b] V. Paxson, “End-to-End Routing Behavior in the Internet,” *IEEE/ACM Transactions on Networking*, 5(5) (October 1997). pp. 601-615.

[Perl92] R. Perlman, *Interconnections: Bridges and Routers*, Addison-Wesley, Reading, MA (1992).

[Post81] J. Postel, *Transmission Control Protocol*, RFC 793 (September 1981).

[PoVa98] P.A. Porras, A. Valdes, “Live Traffic Analysis of TCP/IP Gateways,” *Proceedings of the 1998 Symposium on Network and Distributed System Security* (March 1998). pp. 142-154.

[QVWN+98] D. Qu, B.M. Vetter, F. Wang, R. Narayan, S.F. Wu, Y.F. Jou, F. Gong, C. Sargor, “Statistical Anomaly Detection for Link-State Routing Protocols,” *Proceedings of the 1998 International Conference on Network Protocols* (October 1998). pp. 62-70.

[SCCD+96] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, “GrIDS -- A Graph-Based Intrusion Detection System for Large Networks,” In *Proceedings of the 19th National Information Systems Security Conference* (1996).

[Seat00a] Los Angeles Times and the Associated Press, *Vandals still weaving web of sabotage on the Internet*, newspaper article, The Seattle Times (February 9, 2000).

[Seat00b] S.P. Chan, *Attacks 'only going to get worse,' local experts warn*, newspaper article, The Seattle Times (February 10, 2000).

[SiKe97] K.E. Sirois, S.T. Kent, “Securing the Nimrod Routing Architecture,” In *Proceedings of the 1997 Symposium on Network and Distributed System Security* (February 1997).

[Stee95] M. Steenstrup, *Routing in Communications Networks*, Prentice Hall, Englewood Cliffs, NJ (1995).

[Vene96] W. Venema, “Murphy’s law and computer security,” *Proceedings of the Sixth USENIX Security Symposium* (July 1996). pp. 187-193.

[VLRS99] S. Vimercati, P. Lincoln, L. Ricciulli, P. Samarati, “PGRIP: PNNI Global Routing Infrastructure Protection,” *Proceedings of the 1999 Symposium on Network and Distributed System Security* (February 1999). pp. 135-149.

[Wash00] *Vandals Cripple 3 More Web Sites; Reno Orders Probe; DOW Drops 258.44*, newspaper article, The Washington Times (February 10, 2000).

[ZCD97] E.W. Zegura, K.L. Calvert, M.J. Donahoo, “A Quantative Comparison of Graph-Based Models for Internet Topology,” *IEEE/ACM Transactions on Networking*, 5(6) (December 1997). pp. 770-783.

[Zhan98] K. Zhang, “Efficient Protocols for Signing Routing Messages,” *Proceedings of the 1998 Symposium on Network and Distributed System Security* (March 1998). pp. 29-35.

7 Appendix: WATCHERS Implementation Details

This Appendix provides a loosely structured account of design, compilation, and execution details of the WATCHERS implementation.

7.1 Design

In anticipation that no in-transit fragmentation would occur in the router testbed of Figure 4.2, all WATCHERS counters were made to be *per-packet*, not *per-byte* (See sections 3.1 and 5.2.4). *T*, *S*, and *D* counters were split into two tables each, one for the inbound traffic, and one for outbound.

All WATCHERS counters were stored in arrays indexed by an internal *router table* unique to each router. Assuming no changes to the topology would occur during run-time, these tables were stored in configuration files read once upon initialization. Router tables always listed the host router as the first entry, followed by immediate neighbors, then their neighbors, and then other routers in the AS (See Figure 4.1 for example). No support was included for *external nodes* (See section 3.1).

This router table indexing strategy was chosen to simplify counter storage and exchange, at the expense of remote manipulation during the diagnosis phase (See section 3.2.2). Because each router table was unique, every router needed to know the router tables for each of its neighbors, and each of their neighbors.

In the interest of rapid development, the entire set of each router's WATCHERS counters (excluding *M* counters) and its router table were passed during counter exchange, and without regard to byte order (the x86 machines used during testing all store variables using *little endian*, or Least Significant Byte first).

Only a few optimizations were undertaken in the source code; these included elimination of spin-wait loops to improve CPU performance, manual parsing of *tcpdump* output, and improved synchronization using more accurate timers. The *ncurses* library was used to create a portable text-windowed user interface (See Figure 4.1).

7.2 Compilation

The WATCHERS source code was written in C and comprised roughly 6,000 lines of code at the time of this writing. Separate modules were defined for high- and low-level counter and message manipulation, router index table maintenance, timers, and user interface routines.

WATCHERS and its modules were compiled using *gcc* version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release) with the flags "*–O3 –Wall*". Any reported errors and warnings were corrected prior to running the tests in section 4.3.

7.3 Execution

Except where specified otherwise, WATCHERS was run by *root* on the command line at normal priority. An attempt was made to minimize CPU utilization during tests; when queried, the machines reported near-zero load at all times, even during WATCHERS execution.

8 Appendix: Network Configuration

100MB Ethernet category 5 crossover cables were used to connect pairs of router-configured machines as in Figure 4.2. In Table 8.1 below, information is shown for each of the machines in the testbed. Right and left interfaces refer to the clockwise and counter-clockwise directions in Figure 4.2. Netmasks were assigned to 255.255.255.0 for each Ethernet interface.

Host Name	*rushmore*	*whistler*	*shasta*	*washington*	*norquay*	*baker*
OS	RedHat Linux release 6.2 (full installation)					
Kernel	2.2.14-5.0	2.2.14-5.0	2.2.14-5.0smp	2.2.14-5.0	2.2.14-5.0	2.2.14-5.0
Processor Type	i686					
Processor Name	Pentium II	Pentium III	Pentium Pro	Pentium III	Pentium III	Pentium III
Stepping	2	3	9	3	3	3
CPU MHz	448.061111	451.029699	199.435738	551.260463	551.258695	551.247374
Cache Size (KB)	512	512	256	512	512	512
Bogomips	447.28	448.92	199.07	548.86	550.50	548.86
Total Memory (KB)	257684	127916	62992	257684	257684	257684
Memory Used (%)	31	60	95	30	30	33
Total Swap Space (KB)	72252	72252	80284	72252	80284	72252
Swap Space Used (%)	0	0	4	0	0	0
Root Filesystem Capacity (MB)	8485	19171	1914	19178	19171	19171
Root Filesystem Space Used (%)	21	47	76	12	12	12
Left Ethernet Card	Intel Ether-Express PRO/100 PCI WOL	Built-in	Intel Ether-Express PRO/100 PCI WOL	Built-in	Built-in	Built-in
Right Ethernet Card	Linksys EtherFast 10/100 PCI	3Com EtherLink 10/100 PCI	3Com EtherLink 10/100 PCI	Linksys EtherFast 10/100 PCI	Linksys EtherFast 10/100 PCI	Linksys EtherFast 10/100 PCI
Left Ethernet Driver	eepro100	3c59x	eepro100	3c59x	3c59x	3c59x
Right Ethernet Driver	tulip	3c90x	3c90x	tulip	tulip	tulip
Left IP Address	10.0.12.1	10.0.11.1	10.0.10.1	10.0.9.1	10.0.14.1	10.0.13.1
Right IP Address	10.0.11.10	10.0.10.10	10.0.9.10	10.0.14.10	10.0.13.10	10.0.12.10

Table 8.1: Selected configuration parameters as reported by *Linuxconf* and GNOME's *System Information* tool, and manual inspection of the hardware.