Detecting Disruptive Routers: A Distributed Network Monitoring Approach

Kirk A. Bradley* Steven Cheung Nick Puketza Biswanath Mukherjee Ronald A. Olsson
Department of Computer Science
University of California, Davis
Davis, CA 95616
{bradley,cheung,puketza,mukherje,olsson}@cs.ucdavis.edu
http://www.cs.ucdavis.edu

Abstract

An attractive target for a computer system attacker is the router. An attacker in control of a router can disrupt communication by dropping or misrouting packets passing through the router. We present a protocol called WATCHERS that detects and reacts to routers that drop or misroute packets. WATCHERS is based on the principle of conservation of flow in a network: all data bytes sent into a node, and not destined for that node, are expected to exit the node. WATCHERS tracks this flow, and detects routers that violate the conservation principle. We show that WATCHERS has several advantages over existing network monitoring techniques. We argue that WATCHERS' impact on router performance and WATCHERS' memory requirements are reasonable for many environments. We demonstrate that in ideal conditions WATCHERS makes no false-positive diagnoses. We also describe how WATCHERS can be tuned to perform nearly as well in realistic conditions.

1 Introduction

The router is a primary component in the infrastructure of today's Internet, and is therefore an attractive target for attackers. If an attacker can gain control of a router, the attacker can disrupt communication by dropping or misrouting packets passing through the router. We present a protocol that detects and reacts to routers that drop or misroute packets.

The protocol is called WATCHERS: Watching for Anomalies in Transit Conservation: a Heuristic for Ensuring Router Security. WATCHERS protects the routers in an *autonomous system* (AS), a set of routers and networks controlled by one administrative authority. WATCHERS is *distributed*: each participating router concurrently runs the WATCHERS algorithm. Each router checks incoming packets to see if they have been routed correctly. Also, each router counts the data bytes that pass through neighboring routers. Periodically, the routers report their counter values to one another, and each router checks if any of its neighbors have violated the *principle of conservation of flow*. This principle asserts that all data bytes sent into a node, and not destined for that node, are expected to exit the node. When a router finds a neighbor that violates the principle, or a neighbor that is misrouting packets, the router stops sending packets to that neighbor. Eventually, the bad router is effectively removed from the network, because all of the bad router's neighbors stop sending packets to it.

WATCHERS has four significant advantages over other network monitoring techniques:

- A network monitoring tool (e.g., *traceroute* [5] or an implementation of the Simple Network Management Protocol (SNMP) [4]) may fail to detect an attack because the attacker is able to disrupt messages sent by the tool, including messages between separate tool components. WATCHERS

*Kirk Bradley's current affiliation is SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493 (e-mail: bradley@systech.sri.com). This work has been supported by the National Security Agency INFOSEC University Research Program under Contract No. DOD-MDA904-96-1-0118, and by the Defense Advanced Research Projects Agency under grant ARMY/DAAH 04-96-1-0207.

uses *flooded transmissions* (see Section 1.1) and message authentication to prevent attackers from interfering with communication.

- WATCHERS can detect routers that *selectively* drop or misroute packets, as well as routers that *cooperate* to conceal malicious behavior.
- When they detect suspicious behavior, most network monitoring techniques are unable to locate the malicious (or faulty) routers, or they are only able to identify a list of potential suspects [4, 5, 8, 13]. WATCHERS can identify the exact router(s) which are dropping or misrouting packets.
- In ideal conditions we show that WATCHERS never identifies a good router as bad (i.e., never makes a *false-positive* diagnosis). We also show how WATCHERS can be tuned to perform nearly as well in realistic conditions.

The rest of this section provides background information on (1) our model of an AS, (2) routing, (3) malicious router behavior, and (4) current router monitoring techniques and their limitations[1]. Section 2 presents the WATCHERS protocol. Section 3 describes future research tasks. The Appendix proves that WATCHERS is *correct*: WATCHERS does not make any false-positive diagnoses (when certain conditions hold). For more details on this work, consult the comprehensive report [3].

1.1 Our Model and Associated Terminology

An *internetwork* is a group of networks connected so that computers in different networks can communicate. The networks are connected by routers. Each communication between two computers across a network (or internetwork) is divided into segments of data called *packets*. When a packet must travel from one network to another, it is sent to a router. The router receives the packet from one network, and forwards the packet to another network closer to the packet's destination. Each router uses a *routing table*, a matrix that indicates where to forward a packet based on the packet's destination. Each entry in the routing table maps a destination address to the address of the first router on the shortest path to that destination.

The routers and networks in an AS comprise an internetwork. We model an AS as a *directed graph* in which nodes represent routers and edges represent communication links. (Thus a physical link should be represented by two edges with opposite directions. For clarity, though, we will often represent a link with a single bi-directional edge.) Routers at either end of a link are called *neighbors*. When a packet travels from one computer to another, it traverses a sequence of zero or more routers (a *route*). A *hop* is a traversal of one link from one router to another.

A *flooded transmission* is one in which the data is sent over every link in the network (reaching every node). Flooding is fast and is the most robust form of communication in a network containing faulty nodes [13].

1.2 Routing

The routers in an AS communicate to update their routing tables as the network topology changes according to a *routing protocol*. Each router in an AS runs the same routing protocol. Many routing protocols fit the following description. Each router monitors the network for topology changes (e.g., temporarily inoperative links) and reports changes to the other routers in the AS via update messages. When a router receives an update message, it recomputes the shortest path to each other router in the AS, and adjusts its routing table accordingly.

1.3 Malicious Router Behavior

WATCHERS detects routers that drop packets and/or misroute packets. We refer to such routers as *bad routers*, and we more specifically refer to a router that drop packets as a *network sink*. The simplest network sink drops all packets. It behaves just like an inoperable router, so it can be detected using existing methods (e.g., *traceroute*). However, an *intelligent network sink* drops packets selectively. Some drop packets periodically, while others drop packets based on their content, including the source and destination address. Still other network sinks consort with other routers to conceal evidence of packet-dropping.

Intelligent network sinks are difficult to detect, since neither the source nor the destination is notified of the location where the packet was lost. Often, the source does not even know the route traversed by the packet before it was lost. Thus, the source cannot pinpoint the location of the network sink.

1.4 Current Router Monitoring Techniques

Many router monitoring techniques already exist. Here we summarize several of them and comment on their abilities to detect bad routers.

1.4.1 Hop-by-Hop Acknowledgements

Perlman first proposed the idea of intermediate hop acknowledgments [13]. Under this scheme, when a

[1] Readers familiar with these topics may wish to skip to Section 2 (our WATCHERS protocol).

host S sends a packet to host D, S receives an acknowledgement (ACK) from D *and* from each router on the path from S to D.

If a packet is dropped on the path from S to D, it appears that the bad router must be either the first router on the path which did not send an ACK to S, or the router previous to that router in the path.

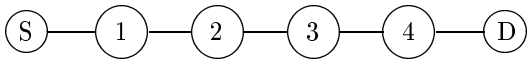

Figure 1: A path of four routers between two hosts.

However, this security mechanism can be defeated by *consorting bad routers* (routers that cooperate to hide malicious behavior) [13]. In Figure 1, suppose router 4 drops a packet (from S and intended for D) while router 1 covers for router 4 by dropping the ACK from router 3. Then it *appears* to S that either router 2 dropped the packet (but sent an ACK) or router 3 dropped the packet (and sent no ACK). Router 1 has shifted the blame away from router 4.

1.4.2 Protection from Packet Corruption

A router can alter a packet's contents to cause the packet to be dropped or misrouted. For example, if a packet's destination address is changed in transit, that packet will be misrouted. Such alterations can be prevented by checksums and cryptographic authentication mechanisms. However, these measures do not protect a packet from being dropped.

A hop-by-hop checksum verification has been proposed [13]. The checksum is verified at each router to isolate a packet-corrupting router. This method is both time-intensive and open to attack. The simplest way to defeat this method is to corrupt the packet and regenerate the checksum. Hence, it is not a realistic solution for malicious router problems.

1.4.3 A Probing Technique

Another way to detect misbehaving routers is a probing technique based on peer testing of routers [6]. Using this strategy, a router (the *testing router*) can test its peers along a suspicious route. The testing router sends a *probe* to the first router on the route. A probe is a packet that will follow a pre-determined path, ending back at the testing router. If the probe returns, the next probe is sent through the first (already-tested) router to the second router. If this second probe returns, the next probe is sent through the first and second routers to the third router, and so on until a probe does not return or the destination is reached. If a probe does not return, the router farthest away from the testing router along the probe path is considered faulty or malicious.

While probing should be useful for locating faulty routers, probing does have a weakness: A malicious router can avoid detection if it can distinguish testing probes from ordinary packets.

1.4.4 Network Management

A network management tool monitors the routers and networks in an AS, and can be used to debug problems, control routing, and find computers that violate protocol standards [4, 7]. Unfortunately, current network management tools (e.g., implementations of SNMP) cannot be used to monitor all of the traffic in a network [2, 4, 9], and therefore they cannot be used to detect all bad routers.

1.4.5 Recording and Tracing Routes

Route recording and *route tracing* are often the first methods used to isolate faults in routers [5]. For the Internet Protocol (IP), route recording is optional for each packet. When this option is selected, each router on the path to the packet's destination appends its IP address to a list in the packet's header. However, route recording is a poor way to isolate malicious routers for three reasons. First, the source and destination must agree in advance to use the recording mechanism; if the destination is not informed, it will disregard the route list. Second, the list can be altered by an intermediate router. Third, an intermediate router can simply drop the packet, so that the route list never reaches the destination [7].

Route tracing is similar to probing. The main difference is that probing tests the routers on a pre-selected path between two hosts; route tracing identifies and tests the routers in the path that a packet would actually take between the hosts. Thus, route tracing reveals the routing decisions of the intermediate routers, and so the method can be used to detect a router that misroutes packets. (The *traceroute* program [5] is a popular implementation of route tracing.)

Route tracing suffers from the same limitation as probing; a bad router might be able to identify a test packet, and thereby avoid detection. Furthermore, since route tracing tests one route at a time, and the number of possible routes in an AS is often large, it is usually impractical to use route tracing to continuously monitor all the routers in AS.

To summarize, existing router monitoring techniques are not adequate for detecting bad routers. Instead, it is necessary to monitor packet flow to detect routers that drop and/or misroute packets. This is the technique that WATCHERS implements.

However, WATCHERS will not detect all malicious behavior in routers, because routers can misbehave in many other ways. A router can send false topology update messages that affect the routing tables of routers throughout the network. Multiple problems may result, including the routing of packets to compromised routers, sub-optimum routes through the network, and isolated routers (i.e., routers that do not receive any packets from their neighbors). A router may also alter or inspect the packets passing through it, or inject packets spuriously into the network. Detailed examples of other malicious router behaviors appear in references [1] and [11].

2 The WATCHERS Protocol

This section describes the details of WATCHERS. The goal of WATCHERS is to identify bad routers. Section 1.3 defines a bad router as a router that drops or misroutes packets; here we extend the definition to include routers that refuse to participate in the WATCHERS protocol.

2.1 Conditions

The following conditions are necessary for WATCHERS to work correctly.

1. *Link-State Condition:* The routers must use a *link-state* routing protocol. In such a protocol, each router is aware of each other router and each link between pairs of routers in the AS. Also, each router periodically broadcasts an update message to indicate which of its links to other routers are "up" and which are "down" [7].

2. *Good Neighbor Condition:* Every router must be directly connected to at least one good router.

3. *Good Path Condition:* Each pair of good routers must be connected by at least one path of good routers. From this condition and the Good Neighbor Condition, it follows that when a router floods a message, it reaches every good router in the AS.

4. *Majority Good Condition:* Good routers must be in the majority. This condition prevents a group of bad routers from triggering the start of a new round of WATCHERS (see Section 2.3.1).

2.2 WATCHERS Counters

Each router maintains counters for *transit traffic* and *misrouted traffic*.

2.2.1 Counting Transit Packets

Henceforth we will refer to the first router in a packet's route as the packet's source, and to the last router in a packet's route as the packet's destination. A packet is a *transit packet* for a router if the packet passes through the router, but the router is neither the packet's source nor its destination. For any pair of routers, we identify three types of transit packets for each direction of traffic. Each router counts the number of data bytes in each transit packet, using a separate counter for each type. Figure 2 illustrates the three types of transit packets and the associated counters. Consider traffic flowing from X to Y. The $T_{X,Y}$ counter refers to packets that pass through both X and Y. The $S_{X,Y}$ counter refers to packets with source X that pass through Y. The $D_{X,Y}$ counter refers to packets with destination Y that pass through X. The other three counters are similar, but for the opposite direction of flow. Routers X and Y both maintain copies of all six counters to check each other during the WATCHERS diagnosis phase.

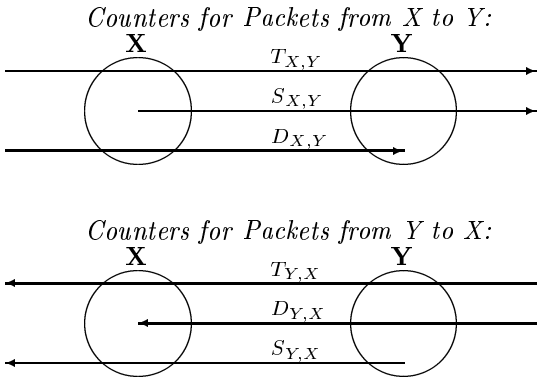

Figure 2: Transit packet byte counters.

2.2.2 Counting Misrouted Packets

Each router also counts the number of times each of its neighbors misroutes traffic. For each neighbor X of router Y, the counter $M_{X,Y}$ records the number of times router X wrongly, with respect to the optimal path, forwards (*misroutes*) a packet to Y. To detect such misrouting, each router maintains a copy of each neighbor's routing table. If an M-counter value is larger than an acceptable threshold (possibly zero), then WATCHERS identifies the associated neighbor as a bad router during the WATCHERS diagnosis phase.

2.3 Communication and Diagnosis in WATCHERS

Each router executes the WATCHERS protocol at regular intervals; each execution of WATCHERS is a *round*. In each round, each participating router in the AS runs the protocol concurrently. Each round has two components. First, the *request, receive, and respond sub-protocol* is used to synchronize the routers and to exchange counter values among the routers. Second, each router runs the *validation* and *conservation-of-flow* algorithms to diagnose neighboring routers as good or bad.

2.3.1 Request, Receive, and Respond

A router participates in the request, receive, and respond (RRR) sub-protocol as follows. First, the router floods a synchronization message (called a *request message*) to indicate a desire to start a new round of WATCHERS. Each request message includes a digitally-signed message digest so that the receiver can authenticate the message's source and contents. The router then waits until it has received a request message from a majority of the routers in the AS. Next the router records a "snapshot" of its counter values, and floods a message containing the values (called a *response message*) to the other routers. Like the request messages, each response message must include a digitally-signed message digest. Neighbors that do not send response messages will be identified as bad during the WATCHERS diagnosis phase. When a router has received all of the counter values it needs, it begins the diagnosis phase of WATCHERS, explained in Section 2.3.2.

Typically, a router starts executing the RRR protocol when its clock indicates that it is time to start a new round of WATCHERS. However, a router will also start the RRR protocol when it receives a request message from a majority of the other routers in the AS. Thus, a router with a malfunctioning clock can still synchronize itself with the other routers.

2.3.2 Diagnosis

The WATCHERS diagnosis algorithm appears in Figure 3. The algorithm has two parts: *validation* and *conservation-of-flow analysis*.

Terminology and Concepts
A *testing router* is a router running the diagnosis algorithm to test its neighbors (the *tested routers*). A *shared link* is the link between a testing router and a tested router. To test another router, a testing router needs to use its own counters, the tested router's counters, and the counters from each of the tested router's neighbors.

Validation
During validation, a testing router examines its neighbors' counters. Each neighbor's counters for the shared link should match the testing router's corresponding counters. A discrepancy between any pair of corresponding counters that is larger than a predetermined threshold (possibly zero) indicates that the tested router is bad.

Each testing router must also check if each neighbor's counters match the counters of *their* neighbors. If there are discrepancies, then the testing router will not perform the next stage of diagnosis (conservation-of-flow analysis) on that neighbor. Instead, the testing router can assume that the neighbor will fail at least one of the validation tests administered by the neighbor's neighbors.

Validation cannot locate all bad routers. Validation identifies routers that drop packets and then change their counters to conceal their behavior. To find bad routers that drop packets but leave their counters intact, we use conservation-of-flow analysis.

Conservation-of-Flow Analysis
In the second stage of diagnosis, a testing router performs conservation-of-flow analysis on each neighbor. For example, if router X in Figure 4 is the *testing router*, it performs a test on routers A and B, its neighbors. To test the flow through A, X needs counters from A and each of A's neighbors: C, E, and X itself. To test the flow through B, X needs counters from B, C, F, and X itself.

Conservation-of-flow analysis is based on a simple principle: in any time interval, the number of data bytes going into a router (less the number of bytes destined for the router) should match the number of data bytes that come out of the router (less the number of bytes sourced by the router). The former quantity is the *incoming transit flow* and the latter quantity is the *outgoing transit flow*.

For example, suppose X, in Figure 4, wants to test A. X can calculate the *incoming transit flow* by using the counters for A's incoming edges (i.e., the counters tracking flow from routers X, C, and E). The expression for the incoming transit flow (I) is:

$$I = \sum_{\forall N | A \leftrightarrow N} (S_{N,A} + T_{N,A})$$

The summation does not include $D_{N,A}$, since this counter is for data bytes destined for A.

Algorithm 1 (Diagnosis algorithm, for a given router r)

For each $d \in$ *destinations in the AS*
    For each n *such that* $r \leftrightarrow n$
        / Local validation of r's outgoing edges */*
        if *(n's respond message has been received and authenticated)* and
            $(r.T_{r,n}[d] = n.T_{r,n}[d] \wedge r.S_{r,n}[d] = n.S_{r,n}[d] \wedge r.D_{r,n}[d] = n.D_{r,n}[d])$
        then
            / Compare neighbor's counters to its neighbors' counters */*
            if $(\forall t \in \{x | x \leftrightarrow n\} \cdot$
                (t's respond message has been received and authenticated) and
                $(n.T_{n,t}[d] = t.T_{n,t}[d] \wedge n.S_{n,t}[d] = t.S_{n,t}[d] \wedge n.D_{n,t}[d] = t.D_{n,t}[d])$ and
                $(n.T_{t,n}[d] = t.T_{t,n}[d] \wedge n.S_{t,n}[d] = t.S_{t,n}[d] \wedge n.D_{t,n}[d] = t.D_{t,n}[d]))$
            then
                / Conservation-of-flow analysis */*
                if $\sum_{\forall t | t \leftrightarrow n}(n.T_{t,n}[d],\ n.S_{t,n}[d]) \neq \sum_{\forall t | t \leftrightarrow n}(n.T_{n,t}[d],\ n.D_{n,t}[d])$
                then r *diagnoses* n *as bad;*
            else *do nothing (other routers will diagnose the problem);*
        else r *diagnoses* n *as bad;*
    end

    For each n *such that* $r \leftrightarrow n$
        / Local validation of r's incoming edges */*
        if *(n's respond message has NOT been received and authenticated)* or $(r.M_{n,r} \neq 0)$ or
            $(r.T_{n,r}[d] \neq n.T_{n,r}[d] \vee r.S_{n,r}[d] \neq n.S_{n,r}[d] \vee r.D_{n,r}[d] \neq n.D_{n,r}[d])$
        then
            r *diagnoses* n *as bad;*
    end
end

Figure 3: WATCHERS Diagnosis algorithm. (Note: The character before the dot in each counter reference identifies the counter's owner.)

Similarly, the expression for the outgoing transit flow (O) is:

$$O = \sum_{\forall N | A \leftrightarrow N} (D_{A,N} + T_{A,N})$$

The counters needed to compute the incoming transit flow and the outgoing transit flow are illustrated in Figure 5. Now, to test A for conservation of flow, X simply compares A's incoming transit flow to its outgoing transit flow:

if $|I - O| >$ **threshold**
then X diagnoses A as a network sink.

2.3.3 Destination-Specific Counters

The counters described in Section 2.2.1 have one problem: they do not provide enough information to detect consorting routers (see Section 1.4.1).

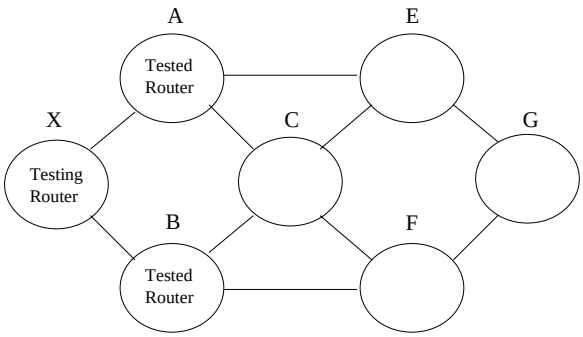

Figure 4: A sample router configuration labeled according to testing terminology.

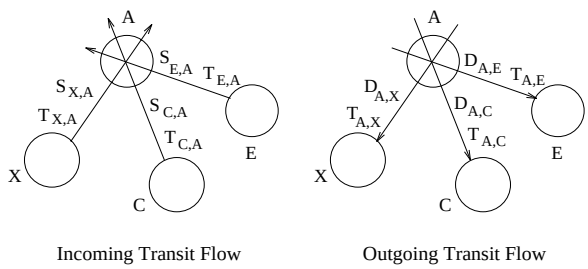

Figure 5: Incoming transit flow vs. outgoing transit flow: X testing A from Figure 4.

For example, suppose that routers 3 and 4 in Figure 6 are consorting to drop packets. Assume that the correct route from A to B is $A \longrightarrow 1 \longrightarrow 3 \longrightarrow 4 \longrightarrow B$. However, when A sends a packet to B, the packet travels through routers 1 and 3 (correctly), but then router 4 drops the packet instead of forwarding it to B.

Routers 3 and 4 can hide this attack by incrementing their $D_{3,4}$ counters (instead of their $T_{3,4}$ counters) when the packet reaches router 4. Then conservation-of-flow analysis will not detect the attack.

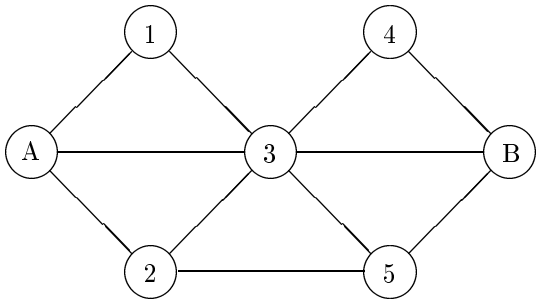

Figure 6: A sample router configuration.

To solve this problem, we need to increase the logging requirements. Instead of two S counters and two T counters per neighbor, each router must maintain two S counters and two T counters per neighbor *per destination*. For example, given Figure 6, suppose that a packet travels from router A to router B along this route: $A \longrightarrow 2 \longrightarrow 3 \longrightarrow 5 \longrightarrow B$. When router 2 sends the packet to router 3, both routers 2 and 3 should increment their $T_{2,3}[B]$ counters. The $T_{2,3}[B]$ counter is for packets *with destination* B that pass through router 2 and then pass through router 3.

Then, instead of one conservation-of-flow check per neighbor, each router must perform one check per neighbor per destination. The flow of data bytes *with a particular destination* into a router must match the flow of data bytes *with that same destination* out of the router.

Consider the attack example again, in which router A sends a packet to router B, and the correct packet route is $A \longrightarrow 1 \longrightarrow 3 \longrightarrow 4 \longrightarrow B$. When router 1 sends the packet to router 3, both routers must increment their $T_{1,3}[B]$ counters. When router 3 forwards the packet to router 4, router 4 drops the packet, and both routers 3 and 4 increment their $D_{3,4}$ counters to attempt to hide the attack. However, when router 1 checks the flow of data bytes *with destination* B into and out of router 3, it will find less data bytes coming out than went in. Thus, using this new strategy, router 1 *can* detect router 3's bad behavior.

The ability to detect such an attack by consorting routers is an improvement over the conservation-of-flow analysis technique as it first appeared in reference [6].

2.4 Response

During a round of WATCHERS, a testing router can diagnose a neighbor as a bad router for four reasons:

1. the testing router did not receive a response message from the neighbor during the RRR subprotocol;
2. the neighbor misrouted a packet;
3. the neighbor failed the validation test; or
4. the neighbor failed the conservation-of-flow test.

For all four cases, the testing router takes the same actions. It floods a routing update advertising the shared link as down (inoperable), which signals the other routers in the AS to remove that link from their network maps. (The bad router may continue to advertise the link as up. However, in at least one link-state routing protocol, the Open Shortest Path First protocol, when two neighbors advertise conflicting information about the status of their shared link, the other routers treat the link as down [12].) Also, the testing router ceases to send packets along that link, and acknowledge traffic received on that link. After all of a bad router's neighbors take these steps, then the bad router is *logically disconnected* from the network.

2.5 Thresholds

Ideally, the transit counters of neighboring good routers should match perfectly, and for each good router the corresponding misrouting counter values

should be zero. In practice, though, the counter values are likely to be different than the ideal values, even for good routers. Therefore, WATCHERS accepts a counter value that is different than the ideal value if the difference is less than a pre-selected threshold.

Three reasons why counter values may be different from the ideal or expected values are:

- We have implicitly assumed that good routers never drop packets. However, typical network protocols, including several within the TCP/IP suite, drop packets due to errors in transmission or congestion in the network [7]. Routers with extremely heavy traffic may drop packets for these reasons at significant rates.
- Neighbors may not be perfectly synchronized when a new round of WATCHERS begins. Assume that there are N routers in the AS, and let $k = \lceil N/2 \rceil$. According to the RRR sub-protocol, each router records a snapshot of its counter values as soon as the kth request messages arrives. We call this instant the *recording time*. The difference in recording times for two neighbors should be less than or equal to the delay on their shared link, since flooding is used to distribute request messages. However, any difference in recording times may cause the counter values of neighbors to disagree.
- Our misrouting detection method depends on neighboring good routers to agree on the network topology. Since a good router sends routing updates whenever it changes its own routing table, and its good neighbors immediately change their own tables accordingly, any period of discrepancy between the routing tables of neighboring good routers should be brief [10]. Moreover, in modern link-state routing protocols, link costs are static and therefore link-state changes are infrequent [15]. Even so, a momentary disagreement over topology may cause a router to increment an M counter with the mistaken belief that a neighbor has misrouted a packet.

The threshold values are also dependent on the WATCHERS *period* (time between consecutive rounds). The period should be adjusted to fit the environment. The period must be at least as long as the time needed by the slowest router to execute one round of WATCHERS. The period should be short enough to avoid counter overflow. Finally, since the threshold values depend on the period, the period and thresholds should be adjusted together.

Setting the thresholds correctly is critical. If the threshold values are too small, too many false alarms may occur. If they are too large, too many attacks might escape detection. However, in some environments, choosing thresholds may be difficult, due to rapidly changing conditions in the network (e.g., network load). Such environments may require threshold values that change dynamically. While we have identified some of the issues involved in setting thresholds, this is an open research problem. Some related work on this problem appears in the intrusion detection literature [14].

2.6 Memory Requirements and Performance

In many environments, WATCHERS' memory requirements are reasonable. Let R be the number of routers in the AS. (Note that R is then also the number of destinations, according to our definition of destination in Section 2.2.1.) In the worst case, a router stores $4 \times N \times R + 3 \times N$ counters, where N is the number of neighbors the router has. To elaborate, each router stores two S counters and two T counters per neighbor per destination ($4 \times N \times R$ counters), two D counters for each neighbor ($2 \times N$ counters), and one M counter for each neighbor (N more counters). However, often a router will send packets to only a few different destinations during one round of WATCHERS, and therefore will only need to use the counters corresponding to those destinations. Thus, usually a router will need just a small fraction of the maximum number of counters during each round of WATCHERS.

We have so far ignored the issue of a packet with a source and/or destination outside of the AS. WATCHERS can handle such packets by treating the set of all routers outside the AS as a single extra node in the AS graph (the *external node*). Each router in the AS that has a link to any router outside the AS is considered a neighbor of the external node. With these minor modifications, WATCHERS can monitor intra-AS traffic. Since this approach adds only one node to the AS graph, it does not significantly change the memory requirements for WATCHERS.

Next, we divide the performance analysis of WATCHERS into several parts.

- Counter Upkeep — Incrementing the packet counters requires only a few additional (simple) operations per packet. Depending on the data structure used to store the counters (e.g., a linked list), additional operations may be required to locate the appropriate counter.
- Routing Table Computation — Each router must compute routing tables for each of its neighbors whenever the network topology changes. This

must be done quickly so that packets are not mistakenly logged as misrouted. Alternatively, the counting of misrouted packets can be suspended until the new tables are ready. Also, a separate processor can compute the new tables so that routing performance is not affected.

- Flow Analysis — In the worst case, each router must perform one conservation-of-flow check per neighbor per destination. In practice, though, each router will usually do much less work, since it is unlikely that each neighbor will receive packets for each destination during a single round of WATCHERS. Also, the diagnosis phase of WATCHERS can be performed by a separate processor, and need not affect the router performance.

To summarize, the memory requirements and performance cost of WATCHERS both depend on the number of routers in the AS. For an AS with a very large number of routers, the memory requirements may be high, although the performance costs can be defrayed by using separate processors for the computation of routing tables and for diagnosis.

3 Future Work

We can identify several potential future tasks, the first of which is to implement and test WATCHERS. By implementing WATCHERS, we could study its memory requirements and performance costs in detail. We could also investigate the important problem of setting good threshold values for WATCHERS counters.

We expect that WATCHERS can be adapted to monitor the collective behavior of a group of several routers (a *supernode*). Each router incident to the supernode would treat the supernode as a single router while running WATCHERS. We predict that it may be less expensive (in terms of memory and computation) to run WATCHERS in this configuration than it would be to run WATCHERS on each router in the supernode. If the supernode is ever diagnosed as bad, then WATCHERS can be run as usual *inside* the supernode to pinpoint the bad routers.

Finally, WATCHERS' use of message authentication, flooded transmissions, and the validation stage of the WATCHERS diagnosis algorithm all make it difficult for a bad router to escape detection. A valuable future task would be a formal analysis of WATCHERS' *effectiveness* in detecting bad routers, similar to our demonstration of WATCHERS' *correctness* in the Appendix.

Appendix: Proof of WATCHERS' Correctness

We say that a bad router detection mechanism is *correct* if the mechanism has this property: When any router B is diagnosed as bad by a good router G, B is in fact a bad router. We claim that WATCHERS is correct if the conditions in Section 2.1 hold, and the following conditions also hold:

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

Proof:

A good router G can diagnose a neighboring router B as bad for four different reasons. For each case, we show that the diagnosis implies that router B *is* bad.

1. *Missing "response" message*: Suppose that G does not receive a "response" message from B during the RRR sub-protocol. Since B and G are neighbors, it follows from the Perfect Transmission Condition that B must not have sent the message. Thus B is bad since it did not participate correctly in the protocol.

2. *Misrouted packet*: Suppose that G detects that B has misrouted a packet. Then B either disagrees with G on the network topology, or B agrees with G but intentionally misrouted the packet. In the latter case, B is bad because it misrouted the packet, and in the former case, since G is good, B must be bad by the Neighbor Agreement Condition.

3. *Validation test fails*: Suppose that at least one of G's counters associated with the link between G and B disagrees with B's corresponding counter. The disagreement implies that either G or B is lying. Since G is good, B must by lying, and is therefore a bad router.

4. *Conservation-of-flow test fails*: Suppose that G finds that B has violated the conservation-of-flow principle. Two explanations are possible. The first is that at least one of the counter values that G used in conservation-of-flow analysis is incorrect. Let S be that set of values. By the action of the detection algorithm, G does not perform flow analysis on B unless the values in S match B's corresponding counter values. Thus, if at least

one value in S is incorrect, then at least one of B's counters must be incorrect, and so B is a bad router. The second possible explanation is that B is dropping packets (or injecting packets) and is therefore a bad router. □

References

[1] S. M. Bellovin. "Security Problems in the TCP/IP Protocol Suite". *ACM Computer Communication Review*, 19(2):32–48, April 1989.

[2] U. Black. *Network Management Standards.* McGraw-Hill, second edition, 1994.

[3] K. A. Bradley. Detecting Disruptive Routers: A Distributed Network Monitoring Approach. Master's thesis, University of California, Davis, September 1997.

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

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

[6] S. Cheung and K. N. Levitt. "Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection". In *Proc. New Security Paradigms Workshop*, Cumbria, UK, September 1997.

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

[8] S. Garfinkel and G. Spafford. *Practical UNIX and Internet Security.* O'Reilly & Associates, Inc., second edition, 1996.

[9] G. Held. *AN Management with SNMP and RMON.* John Wiley & Sons, 1996.

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

[11] L. Joncheray. "A Simple Active Attack Against TCP". In *Proc. 5th USENIX UNIX Security Symposium*, pages 7–19, Salt Lake City, Utah, June 1995.

[12] John Moy. "OSPF Version 2", July 1997. Internet RFC 2178.

[13] R. Perlman. *Network Layer Protocols with Byzantine Robustness.* PhD thesis, Massachusetts Institute of Technology, October 1988.

[14] B. Soh and T. Dillon. "Setting Optimal Intrusion-Detection Thresholds". *Computers and Security*, 14(7):621–31, 1995.

[15] M. Steenstrup. *Routing in Communications Networks.* Prentice Hall, 1995.