**Title**
Detecting Malicious Routers



**Authors**
Mizrak, Alper
Marzullo, Keith
Savage, Stefan

**Publication Date**
2004-05-24

Peer reviewed

.

# Detecting Malicious Routers

Alper T. Mızrak PhD student, CSE/UCSD
Keith Marzullo Professor, CSE/UCSD
Stefan Savage Assistant Professor, CSE/UCSD
{amizrak, marzullo, savage}@cs.ucsd.edu

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

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

**Key Words:** Internet algorithms, Fault tolerant robust routing, Reliable networks, Malicious routers, Byzantine failures, Specification of an anomalous behavior detector.

# 1 Introduction

This paper addresses part of a simple, yet increasingly important network security problem: how to detect the existence of compromised routers in a network and then remove them from the routing fabric.

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

Moreover, such attacks are not mere theoretical curiosities, but they are actively employed in practice. Attackers have repeatedly demonstrated their ability to compromise routers, through combinations of social engineering and exploitation of weak passwords and latent software vulnerabilities [1, 9, 13]. One network operator recently documented over 5000 compromised routers as well as an underground market for trading access to them [23].

Once a router is compromised an attacker need not modify the router's code base to exploit its capabilities. Current standard command-line interfaces from vendors such as Cisco and Juniper are sufficiently powerful to drop and delay packets, send copies of packets to a third party, or "divert" packets through a third party and back. In fact, several widely published documents provide a standard cookbook for transparently "tunneling" packets from a compromised router through an arbitrary third-party host and back again – effectively amplifying the attacker's abilities, including arbitrary packet sniffing, injection or modification [7, 21]. Such attacks can be extremely difficult to detect manually, and it can be even harder to isolate which particular router or group of routers has been compromised.

The problem of detecting and removing compromised routers can be thought of as an instance of *anomalous behavior-based intrusion detection*. That is, a compromised router can be identified by correct routers when it deviates from exhibiting expected behavior. The problem can be broken into three subproblems:

1. Traffic validation. Traffic information is the basis of detecting anomalous behavior: given traffic entering a part of the network, and an expected behavior of the routers in the network, anomalous behavior is detected when the monitored traffic leaving that part of the network differs significantly from what is expected. However, implementing such validation practically requires tradeoffs between the overhead of monitoring, communication and accuracy.

2. Distributed detection. It is impossible to detect an anomaly at a single router. Any detection requires, synchronizing the collection of traffic information and distributing the results so that anomalous behavior can be detected by sets of correct routers.

3. Response. Once a router, or set of routers, is thought to be faulty, the forwarding tables of correct routers must be changed to avoid using those compromised nodes.

We provide a brief discussion on the first problem, but this paper concentrates primarily on the second problem. Given a reliable traffic validation function, we examine how it can be used to build an anomalous behavior detector for compromised routers. We have developed a formal specification for this detector, with properties similar to those used for traditional failure detectors. Finally, we give two simple protocols that implement our specification and analyze their accuracy, completeness and overhead.

## 2 Related Work

There are two threats posed by a compromised router: the attacker may attack by means of the routing protocol (for example, by sending false advertisements) or by having the router violate the forwarding decisions it should make based on its routing tables. The first kind of attack is often called an *attack on the control plane*, and the second is called an *attack on the data plane*.

The first threat has received, by far, the lion's share of the attention in the research community, perhaps due its potential for catastrophic effects. By issuing false routing advertisements, a compromised router may manipulate how other routers view the network topology, and thereby disrupt service globally. For example, if a router claims that it is directly connected to all possible destinations, it may become a "black hole" for most traffic in the network. While this problem is by no means solved in practice, there has been significant progress towards this end in the research community, beginning with the seminal work of Perlman. In her PhD thesis [16], Perlman described robust flooding algorithms for delivering the key state across any connected network and a means for explicitly signing route advertisements. There have subsequently been a variety of efforts to impart similar guarantees to existing routing protocols with varying levels of cost and protection. Generally, these break down into approaches based on ensuring the authenticity of route updates and those based on detecting inconsistency between route updates [20, 12, 10, 18, 4, 8].

By contrast, the threat posed by subverting the forwarding process has received comparatively little attention. This is surprising since, in many ways this kind of attack presents a wider set of opportunities to the attacker – not only denial-of-service, but also packet sniffing, modification and insertion – and is both trivial to implement (a few lines typed into a command shell) and difficult to detect. This paper focuses entirely on the problem of malicious forwarding.

The earliest work on fault-tolerant forwarding is also due to Perlman [16]. In her PhD thesis, Perlman presented network layer protocols with Byzantine robustness. These results are also summarized in her book [17]. She developed *robust flooding*, a method to deliver a packet reliably to all good routers. This requires a *good path condition*, which states that each pair of nonfaulty routers is connected by at least one path of zero or more nonfaulty routers. Robust flooding was designed to be used for public key distribution and broadcasting *link state packets* (LSP), which is a necessary part of link state protocols. Perlman also developed a novel method for robust routing on top of a link state protocol. In this protocol, the source router first computes a route based on its local database and then sends a digitally signed *route-setup packet* along the chosen route. Each intermediate router on the route verifies the signature and allocates the necessary resources for the data packet. If the source router receives an acknowledgment of route-setup from each intermediate router on the chosen route, then it sends the data packet. The destination router sends back another ack, if the data packet reaches to itself. If the source does not receive this ack for the data packet from the destination, then it detects that the chosen route is not reliable and computes a new route.

Several researchers have subsequently proposed lighter-weight protocols for actively probing the forwarding path to test for consistency with advertised routes. Subramanian et al's Listen protocol [20] does this by comparing TCP Data and Acknowledgment packets to provide evidence that a path is part of end-to-end connectivity, while Padmanabhan and Simon's Secure Traceroute [15] achieves a similar goal using signed probe packets targeting intermediate routers. Both approaches only test for gross connectivity and cannot reveal whether packets have been diverted, modified, created, reordered or selectively dropped.

The approach most similar to our own is the `WATCHERS` protocol, which detects disruptive routers based on a distributed network monitoring approach [5, 2, 11]. However, the `WATCHERS` protocol had many limitations in both its traffic validation mechanism and in its control protocol – many of which were documented by Hughes et al. [11]. Many of these weaknesses arose from the absence of a formal specification, a weak threat model and an excessive requirement for per-router state (bounded only by the total size of the network). In Appendix 7.1, we describe `WATCHERS` in enough detail to understand its strengths and weaknesses, including a previously unreported flaw that allows faulty routers to go undetected.

# 3 System Model

Our work proceeds from an informed, yet abstracted, model of how the network is constructed, the capabilities of the attacker, and the complexities of the traffic validation problem. In this section we describe and motivate the assumptions underlying our model.

## 3.1 Network Model

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

Within this network, we presume that packets are forwarded in a hop-by-hop fashion – each router following the directions of a local forwarding table. As well, we assume that these forwarding tables are updated via a distributed link-state routing protocol such as OSPF or IS-IS. In particular, we depend on the routing protocol to provide each node with a global view of the current network topology. Finally, we also assume the administrative ability to assign and distribute shared keys to sets of nearby routers. This overall model is consistent with the typical construction of large enterprise IP networks or the internal structure of single ISP backbone networks, but is not well-suited for routers that interconnect administrative domains using BGP (a link-vector protocol).

We define a *path* to be a finite sequence $\langle r_1, r_2, \ldots r_n \rangle$ of adjacent routers. Operationally, a path defines a sequence of routers a packet can follow. We call the first router of the path the *source* and the last router its *sink*; together, they are called *terminal routers*. A path might consist of only one router, in which case the source and sink are the same. Terminal routers are leaf routers: they are never in the middle of any path.

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

## 3.2 Threat Model

We assume that attackers can compromise one or more routers in a network and may even compromise sets of adjacent routers as well. In general, we parameterize the strength of the adversary in terms of the maximum number of adjacent routers along a given path that can be compromised.

However, we assume that between any two uncompromised routers that there is sufficient path diversity that the malicious routers do not partition the network. In some sense, this assumption is pedantic since it is impossible to guarantee any network communication across such a partition. Another way to view this constraint is that path diversity between two points in the network is a necessary, but insufficient, condition for tolerating compromised routers. What we are proposing is a set of protocols that offer the sufficiency condition in the presence of the necessary diversity.

Recently, Teixeira et al. [22] empirically measured path diversity in ISP networks and found that multiple paths between pairs of nodes were common. Similarly, many enterprise networks are designed with such diversity in order to mask the impact of link failures. Consequently, we believe that this assumption is reasonable in practice. It is worth noting however, that this diversity usually does not extend to individual

local-area networks – single workstations rarely have multiple paths to their network infrastructure. Consequently, the fate of individual hosts and of the router, to which they are connected, are directly intertwined in practice. So, we assume that terminal routers are not faulty.

Since link state protocols operate by periodically measuring and disseminating information, we assume a synchronous system. The failure of a router is defined in terms of an interval of time, which in practice corresponds with a period of time during which traffic measurements are made. Specifically, a router $r$ is *t-faulty* (that is, *traffic faulty*) with respect to a path segment $\pi$ during $\tau$ if $\pi$ contains $r$ and, during the period of time $\tau$, $r$ exhibits arbitrary behavior with respect to forwarding data that traverses $\pi$. For example, router $r$ can selectively alter, misroute, drop, reorder, or delay the data that flows through $\pi$, and it can fabricate new data to send along $\pi$ such that the packets, if they were valid, would have been routed through $\pi$.

### 3.3 Traffic Validation

A compromised router can make arbitrary alterations to the forwarding behavior of that router. Packets can be dropped, modified, reordered, diverted and so on. However, given the distributed nature of packet forwarding it is not possible for an adversary to perfectly conceal this behavior when it is compared against the observations of its neighboring – non-compromised – neighbor routers. For example, it is not possible for a single compromised router $r_2$ to modify or drop a packet traversing the path $\langle r_1, r_2, r_3, r_4 \rangle$ in an undetectable fashion. An outside observer could query router $r_1$ about the packets it had sent and router $r_3$ about the packets it has received and detect any incongruity.

We call this general process *traffic validation* and such mechanisms are the basic failure detectors in our system. We represent traffic validation mechanisms as a predicate *TV*($\pi$, $info_{r_i}^{\pi,\tau}$, $info_{r_j}^{\pi,\tau}$) where:

- $\pi$ is a path segment $\langle r_1, r_2, \ldots r_x \rangle$ whose traffic is to be validated between routers $r_i$ and $r_j$ where both $r_i$ and $r_j$ are in $\pi$;
- $info_r^{\pi,\tau}$ is some abstract description of the packets that router $r$ forwarded to be routed along $\pi$ over some time interval $\tau$.
- If routers $r_i$ and $r_j$ are not faulty, then *TV*($\pi$, $info_{r_i}^{\pi,\tau}$, $info_{r_j}^{\pi,\tau}$) evaluates to *false* iff $\pi$ contains a router that was faulty in $\pi$ during $\tau$.

If all the packets[1] are monitored, then $TV$ is also transitive, due to the transitive nature of packet forwarding; for any path segment $\pi$, if both *TV*($\pi$, $info_a^{\pi,\tau}$, $info_b^{\pi,\tau}$) and *TV*($\pi$, $info_b^{\pi,\tau}$, $info_c^{\pi,\tau}$) are true then *TV*($\pi$, $info_a^{\pi,\tau}$, $info_c^{\pi,\tau}$) is true.

Implementing a traffic validation mechanism is an engineering problem. For example, the most precise form of $info_r^{\pi,\tau}$ is a complete copy of the packets sent and the time at which each was forwarded. However, the storage requirements to buffer these packets and the bandwidth consumed by resending them, make this approach impractical. In practice, designing this function is a tradeoff between accuracy and overhead.

Similarly, in an idealized network $TV$ might be implemented simply using equality; $info_{r_i}^{\pi,\tau} = info_{r_j}^{\pi,\tau}$. However, real networks occasionally lose packets due to congestion, reorder packets due to internal multiplexing, and corrupt packets due to interface errors. Consequently, $TV$ must be somewhat more sophisticated to accommodate this abnormal, but non-malicious behavior – an inherent tradeoff between the acceptable number of false positives and false negatives.

We have explored and implemented a variety of traffic validation mechanisms, including those based on approximate flow conservation, comparisons between incremental hashes of packet content, and set reconciliation protocols. While detailed descriptions and empirical comparisons of these approaches is

[1]The routers monitoring the path segment can decide on a sampling technique and keep track of only the chosen packets. This method may help decreasing cost significantly.

outside the scope of this paper, we have experience with several mechanisms that are hard to defeat, have few false positives and have acceptable implementation and state overheads.

Since this paper explores the higher-level properties and limitations of systems built upon traffic validation, and not the details of the failure detector itself, for the remainder of the paper we assume that $info_r^{\pi,\tau}$ and $TV$ exist and are perfectly accurate.

# 4 Specification

Since the objective of our protocol is to detect compromised routers, an obvious way to couch this problem is as an implementation of a failure detection: we would define detection in terms of accuracy and completeness [3]). Since this detector is based on evaluating traffic collected over a period of time, we can have the failure detector report pairs $(r, \tau)$, which means that $r$ was suspected as being faulty during the time interval $\tau$. A perfect failure detector would implement the following two properties:

**Accuracy (tentative):** A failure detector is *Accurate* if, whenever a correct router suspects $(r, \tau)$, then $r$ was faulty during $\tau$.

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

Notice that we have specified completeness by modeling it on the *Strong Completeness* property of [3]. We did this because all of the correct routers will need to update their link state routing tables on the basis of the detection.

We would implement this failure detector by collecting traffic information from different points in the router network. The data would be collected and evaluated using the traffic validation function. An individual router's failure detector would either do this evaluation itself or base it on the evaluation of another router.

This approach won't result in a failure detector with the desired properties, however. Consider two adjacent routers $r_1$ and $r_2$. The traffic at $r_1$ claims to have sent 100 packets to $r_2$, which agrees with the upstream traffic into $r_1$. Router $r_2$, however, claims to have received only 20 packets from $r_2$, which is consistent with the downstream traffic. Assuming that link between the two routers is reliable, at least one of the two routers is not telling the truth: $r_1$ could have dropped 80 packets or $r_2$ could have dropped them. A failure detector implemented at $r_1$ or $r_2$ can determine which is faulty in a trivial manner: our specification only constrains the behavior of failure detection by correct routers. So, in this case $r_1$ can detect $r_2$ and $r_2$ can detect $r_1$; if either is in fact a correct router, then that router's failure detection is correct. Any other router, however, can't distinguish between the case of $r_1$ being faulty and of $r_2$ being faulty, and so can only report that one of the two routers is faulty. Hence, we weaken the specification: we have the failure detector return a pair $(\pi, \tau)$ where $\pi$ is a path segment. Since a path segment is being reported, we can also restrict the detection to a process being faulty with respect to traffic being forwarded along $\pi$.

**$a$–Accuracy:** A failure detector is *$a$–Accurate* if, whenever a correct router suspects $(\pi, \tau)$, then $|\pi| \leq a$ and some router $r \in \pi$ was faulty in $\pi$ during $\tau$.

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

From our description so far, $a$ would be 2 or 1 if the router making the detection is either $r_1$ or $r_2$.

Note that this argument assumes that a faulty router can report an incorrect value for the traffic that traversed a path as well as alter this traffic. We will use the term *t-faulty* to indicate a router that alters traffic and the term *p-faulty* to indicate a router that misreports traffic. A *faulty* router is one that is t-faulty, p-faulty or both. As before, we will add the phrase "in $\pi$" to indicate that the faulty behavior is with respect to traffic that transits the path $\pi$. Thus, the $a$–Accuracy requirement can result in a detection if a router is either p-faulty or t-faulty.

Distinguishing between p-faulty and t-faulty is useful because, while it is important to detect routers that are t-faulty, it isn't as critical to detect routers that are only p-faulty. Routers that are only p-faulty are not directly altering the traffic flow. Hence, we weaken $a$–Completeness:

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

Consider a path $\pi$ that contains only routers that are faulty in $\pi$. The information that they record about their traffic cannot be counted on to detect this fact. Thus, the detection of faulty paths will be influenced more by the maximum number of adjacent faulty routers rather than the total number of faulty routers. So, instead of imposing an upper bound on the fraction of routers that can be faulty, we impose an upper bound $bad(k)$ on the number of adjacent faulty routers. For example, if $bad(3)$ holds, then there can be no more than 3 adjacent faulty routers in any path.

Making a $bad(k)$ assumption has an effect on failure detection. Assume that $bad(3)$ holds, and consider a path $\langle r_1, r_2, r_3, r_4, r_5, r_6, r_7 \rangle$ in which $r_3, r_4$ and $r_5$ are faulty. Suppose that over an interval $\tau$, $r_1$ through $r_5$ report having forwarded 100 packets that were to traverse this path, and $r_6$ and $r_7$ reports only 20 such packets. Let $r_1$ obtain these counters. It could be the case that $r_4$ dropped the traffic, and $r_3$ and $r_5$ misreported the traffic in an effort to hide the fact that $r_4$ is faulty. From $r_1$'s point of view, however, something is wrong with either $r_5$ or $r_6$, since their counters indicate that traffic has disappeared. To satisfy $a$–Completeness, $p_1$ needs to detect any possible routers that could have led to this traffic discrepancy. So, the failure detector at $r_1$ could report that the path segment $\langle r_3, r_4, r_5, r_6 \rangle$ contains a faulty router, since if $r_5$ is faulty then $r_3$ and $r_4$ could be as well, given $bad(3)$. That is, we could have the failure detector report a $k + 1$-length path segment to accommodate the fact that $bad(k)$ holds.

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

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

The choice between the two completeness properties is not clear. Using $a$–FC Completeness should lead to more accurate detections but it can leave some t-faulty routers undetected. We discuss this tradeoff more in the context of protocols that implement these two completeness conditions.

This specification has no uniformity-like property: it allows a faulty router to suspect correct routers. In practice, this doesn't pose a problem. In the countermeasure protocols that we have (and the countermeasure protocol of WATCHERS), the only suspicions that elicit a reaction is when a router $r$ suspects a path segment

that is adjacent to $r$. In this case, the countermeasure protocol will cause $r$ not to route traffic along the suspected path segment. A faulty router can already drop packets, and so allowing a faulty router to break links with its neighbor adds no further disadvantage.

In our terminology, Perlman's robust routing protocol was designed assuming a Byzantine failure model. It is $m$–Accurate and $m$–Complete, where $m$ is the maximum length of a path in the network. WATCHERS meets this specification. WATCHERS is 2–Accurate but it is neither $a$–Complete nor $a$–FC Complete for any value of $a$.

# 5 $\Pi_2$: A 2–FC Complete and 2–Accurate Protocol

Since $TV$ is itself accurate and complete, an obvious approach is to have each router collect traffic information over some agreed-upon interval, and then use consensus to have all correct routers agree upon the traffic information. With this information, each router can agree upon which routers might be faulty.

For example, consider the 4-path segment $\pi = \langle r_1, r_2, r_3, r_4 \rangle$ where $r_1$ and $r_4$ are not faulty. Let $info_i^{\pi,\tau}$ be the traffic information that router $r_i$ collects over during the agreed upon time interval $\tau$ for the path segment $\pi$. If at least one of the other routers is t-faulty with respect to $\pi$ during this interval, then $TV(\pi, info_1^{\pi,\tau}, info_4^{\pi,\tau})$ will be false. This implies that for some $i$, $TV(\pi, info_i^{\pi,\tau}, info_{i+1}^{\pi,\tau})$ is false, which means that at least one of routers $r_i$ and $r_{i+1}$ is faulty. Since $info_i^{\pi,\tau}$ and $info_{i+1}^{\pi,\tau}$ were disseminated using consensus, all correct routers will know that at least one of $\{r_i, r_{i+1}\}$ is faulty.

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

If we assume that $bad(k)$ holds, then the minimum value of $x$ satisfying the above constraint is $k+2$. Not all paths need be this long, and so a router also collects information about all paths of which it is a member whose length is less than $k + 2$.

## 5.1 Protocol

The resulting protocol $\Pi_2$ is shown in Figure 1. In this protocol, each router $r$ maintains the current topology $T$ from which it derives its routing table. Each router $r$ also maintains a set of path segments $P_r$ that contain $r$ and that $r$ monitors. A router $r$ runs a thread for each path segment in $P_r$. $P_r$, which is computed from $T$, contains all $(k + 2)-$path segments containing $r$ and all $x-$path segments, $3 \leq x < k + 2$ whose ends are terminal routers.

For each path segment $\pi \in P_r$, $r$ synchronizes with the other routers in $\pi$ and collects information for the same traffic passing through $\pi$ for an agreed-upon interval $\tau$. Periodically, $r$ sends that traffic information to all routers in $\pi$ using consensus. This data is digitally signed to prevent an attack during consensus. We use $[x]_i$ to indicate that $x$ is digitally signed by $i$.

Consider the traffic passing through a path segment $\pi$. The traffic will be consistent — that is, $TV(\pi, info_i^{\pi,\tau}, info_{i+1}^{\pi,\tau})$ will be *true* — for each pair of routers $\langle i, i + 1 \rangle$ in $\pi$ unless a discrepancy is introduced by a faulty router. In other words, if $TV(\pi, info_i^{\pi,\tau}, info_{i+1}^{\pi,\tau})$ is *false* then at least one of the two routers $i$ or $i + 1$ is faulty. Note that it could either be t-faulty or p-faulty (because it reports traffic information that does not represent the actual traffic that transited during $\tau$. In either case, a correct router $r$ in $\pi$ will put the

2-path segment $\langle i, i+1\rangle$ into $suspect_r^\tau[\pi]$ set, and reliably broadcast the evidence of the failure detection, ($[info_i^{\pi,\tau}]_i$, $[info_{i+1}^{\pi,\tau}]_{i+1}$), where $info_i^{\pi,\tau}$ is digitally signed by router $i$ and $info_{i+1}^{\pi,\tau}$ is digitally signed by router $i+1$. Upon receiving this information, all other correct routers can evaluate $TV(\pi, info_i^{\pi,\tau}, info_{i+1}^{\pi,\tau})$ as *false* and detect the fault on the 2-path segment $\langle i, i+1\rangle$.

$\Pi_2$ is given in Figure 1. In Appendix 7.3, we show that $\Pi_2$ is **2–Accurate** and **2–FC Complete**.

## 5.2 Overhead

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

**Collecting traffic information:** In the worst case, a router has to collect traffic information for each packet it has routed, independent of the size of $P_r$. As mentioned in Section 3.3, this overhead can be reduced.

**Size of $P_r$:** The size of $P_r$ indicates the number of different set of routers with which $r$ synchronizes, maintains traffic information, and exchanges such information using consensus. By construction, $|P_r|$ is $O(k \times R^{k+1})$ where $R$ is the maximum number of links incident on a router. In practice, though, we expect $|P_r|$ to be much smaller. We have examined two network topologies, Sprintlink and EBONE, that were measured by the Rocketfuel project [19] and counted the number of distinct path segments that a router monitors for different values of $k$ in $bad(k)$ assumption.

The Sprintlink network consists of 315 routers and 972 links. On the average, a router has 6.17 links, and the maximum number of links that a router has is 45. In Figure 2(a), the maximum, average and median of $|P_r|$ that a router is incident on and monitors is given for this network. The empirical results are much smaller than the theoretical upper bound $O(k \times 45^{k+1})$. This is because, among other factors, a link state routing protocol chooses only one path between any two routers.

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

Assuming the same weak threat model of `WATCHERS`, it is sufficient for a router, implementing $\Pi_2$, to maintain one counter for each path segment in $|P_r|$. For this topology, assuming $bad(2)$, a router maintains 216 counters on average; and the largest number of counters a router maintains is $2,172$. If instead we have $bad(7)$, these numbers become 758 and 8,073.

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

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

Each router in path segment $\pi$ reaches consensus about the traffic information over $\pi$ during time interval $\tau$. To do so requires digitally signing the traffic information, since otherwise the replication is not high enough for consensus to be solvable. Thus, there is an issue of *key distribution* depending on the cryptographic tools that are used. Finally, there must be enough path connectivity among the routers to support consensus [14]. An open question is whether a weaker protocol, such as Crusader's Agreement [6], can be used in place of Consensus.

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

# 6 $\Pi_{k+2}$: A (k+2)–Complete and (k+2)–Accurate Protocol

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

The idea is to apply $TV$ just for the end nodes of each path segment in $P_r$. For example, consider the 4-path segment $\pi = \langle r_1, r_2, r_3, r_4 \rangle$ where $r_1$ and $r_4$ are not faulty. Let $info_1^{\pi,\tau}$, $info_4^{\pi,\tau}$ be the traffic information that router $r_1$ and $r_4$ collect over during the agreed upon time interval $\tau$. If at least one of the other routers is t-faulty with respect to $\pi$ during this interval, then $TV(\pi, info_1^{\pi,\tau}, info_4^{\pi,\tau})$ will be false. In this case, $r_1$ suspects $\langle r_2, r_3, r_4 \rangle$. Similarly $r_4$ suspects $\langle r_1, r_2, r_3 \rangle$.

We use this observation to construct a $less$ accurate failure detector protocol. Again, the first issue to address is over which paths a router should record information. For this approach each router needs only keep track of each $x$-path segment of which it is one of the end nodes, for some value of $x$. The number of $x$-path segments can grow very quickly with increasing $x$, and so $x$ should be as small as possible. It must be large enough so that any sequence of faulty routers will be surrounded by correct routers, as this is necessary to detect faulty behavior.

If we assume that $bad(k)$ holds, then the minimum value of $x$ satisfying the above constraint is $k + 2$. However, monitoring only $k + 2$-path segments is not sufficient. For example, given that $bad(2)$ holds, consider the 4-path segment $\pi = \langle r_1, r_2, r_3, r_4 \rangle$ where $r_1$ and $r_3$ are correct; and $r_2$ and $r_4$ are faulty. In this case, $r_1$ and $r_4$ monitors $\pi$ but $r_4$ can hide the fact that $r_2$ is t-faulty by simply sending traffic information to $r_1$ such that $TV(\pi, info_1^{\pi,\tau}, info_4^{\pi,\tau})$ holds. If $r_1$ were to instead also monitor the path $\langle r_1, r_2, r_3 \rangle$, then $r_1$ could detect $r_2$'s faulty behavior. So, it is necessary for a router $r$ to monitor all $x$-path segments for $x \leq k + 2$ of which $r$ is an end.[2]

## 6.1 Protocol

Each router $r$ maintains $T$, which is the current topology from which it derives its routing tables. As before, each router $r$ also maintains a set of path segments $P_r$ that r monitors. For this protocol, though, $P_r$ contains all $x-$path segments such that $r$ is one of the end nodes and $3 \leq x \leq k + 2$. This is the smallest set of path segments for which there is at least one path segment that has a correct router at the other end.

For each path segment $\pi \in P_r$, $r$ synchronizes with the other end router of $\pi$ and collect information for the traffic passing through $\pi$ during an agreed-upon interval $\tau$. Router $r$ then exchanges this traffic information with the router $r'$ on the other end. If $r$ then finds $TV(\pi, info_r^{\pi,\tau}, info_{r'}^{\pi,\tau})$ is false then there is at least one faulty router faulty in $\pi$ during $\tau$. In particular, either $r'$ is p-faulty or some router in $\pi$ is t-faulty.

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

$\Pi_{k+2}$ is given in Figure 3. In Appendix 7.4, we show that $\Pi_{k+2}$ is **(k+2)–Accurate** and **(k+2)–Complete**.

## 6.2 Overhead

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

**Collecting traffic information:** Assuming that a router uses the same values of $\tau$ for all the path segments in $P_r$, in the worst case a router has to collect traffic information for each packet it routes, which is independent of the size of $P_r$. The same holds for the previous protocol $\Pi_2$. However, in $\Pi_{k+2}$ this cost can be reduced by using sampling. For each $\pi$ in $P_r$, $r$ can agree with the router $r'$ on the other end on a random sampling pattern. The traffic they record on $\pi$ would be determined by this pattern. Although the

[2] Another reason to monitor these path segments is that not all paths need be $k + 2$ long.

faulty routers in $\pi$ could share their information on sampling and only attack the packets not being sampled by a faulty router, by construction there is a path segment in $P_r$ whose other end is not faulty, and so by using suitable encryption any intermediate faulty routers will not know which packets are being sampled for traffic information. We don't know of a similar method of sampling that could be used for $\Pi_2$.

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

For the same Sprintlink(US) network topology that was analyzed before, the maximum, average and median of $|P_r|$ that a router monitors in this protocol is given in Figure 2(b) for different values of $k$ in *bad*$(k)$ assumption. As expected, these values are much lower than the theoretical upper bound, and are also much lower than the corresponding values for $\Pi_2$.

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

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

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

# 7 Discussion and Conclusion

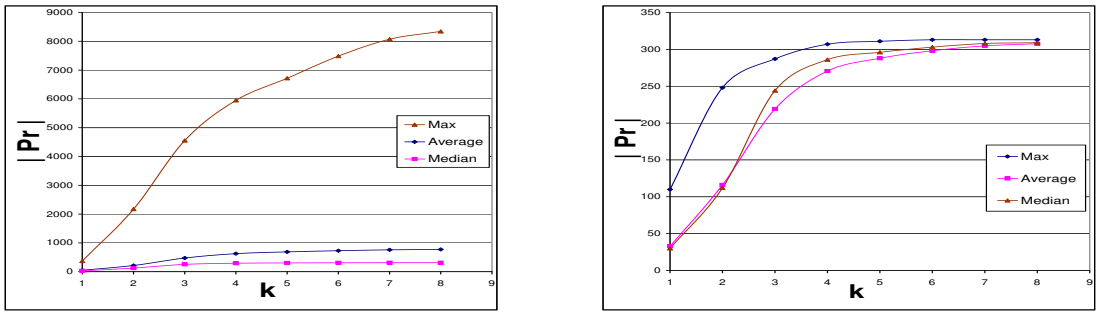
We have given two protocols that represent two different points in a tradeoff among accuracy, completeness, and protocol overhead. The expensive protocol, $\Pi_2$, results in more accurate failure detections but it can leave some t-faulty routers undetected. Meanwhile, with a small overhead, $\Pi_{k+2}$ results in less accurate detections but it detects an unreliable path-segment for each t-faulty router. There are several questions that we have not answered, including the bounds that describe the exact nature of this tradeoff.

We have developed these protocols in conjunction with *countermeasure* protocols that isolate a part of the network that contains compromised routers. We don't discuss the countermeasure protocols here, but these two protocols should be analyzed in conjunction with countermeasure protocols in terms of effectiveness in isolating faulty routers that are introducing discrepancy in to the network traffic. For example, on the surface it appears that we are exploiting a tradeoff between accuracy and protocol overhead when we developed $\Pi_{k+2}$ as a cheaper alternative to $\Pi_2$. In fact, it does not appear that $\Pi_{k+2}$ is less effective in isolating compromised routers or is more likely to partition a network than $\Pi_2$. However, we still don't understand at this point the exact nature of the tradeoff of decreasing accuracy.

We believe that the second protocol, $\Pi_{k+2}$, has a low enough overhead that a practical version could be built. We are currently working on such a protocol and traffic validation function, as well as a practical countermeasure protocol that could be integrated with a link-state protocol with a small extension to the protocol and link update messages.

**failure detector**()
**cobegin**
  **for each** path segment $\pi \in P_r$:
    $suspect_r^\tau[\pi]$ = { } // the set of suspicious path segments in $\pi$ that r detects during $\tau$
    **while** (true) {
      **synchronize** with all routers in $\pi$;
      **collect** traffic information $info_r^{\pi,\tau}$ about $\pi$ for an agreed-upon interval $\tau$;
      **consensus** $([info_1^{\pi,\tau}]_1, [info_2^{\pi,\tau}]_2, ..., [info_{|\pi|}^{\pi,\tau}]_{|\pi|})$;
      // at this point all correct routers in $\pi$ agree on the values of $info_i^{\pi,\tau}$
      **for all** i: $1 \leq i < |\pi|$:
        **if** $\neg TV(\pi, info_i^{\pi,\tau}, info_{i+1}^{\pi,\tau})$ **then**
          $suspect_r^\tau[\pi] = suspect_r^\tau[\pi] \cup \{\langle i, i+1\rangle\}$;
          **reliable broadcast** $([info_i^{\pi,\tau}]_i, [info_{i+1}^{\pi,\tau}]_{i+1})$;
    }
**coend**

Figure 1: $\Pi_2$, A 2–FC Complete and 2–Accurate Protocol




(a) $\Pi_2$ (b) $\Pi_{k+2}$

Figure 2: Based on $bad(k)$; max, average and median size of $P_r$ that a router monitors in $\Pi_2$, and $\Pi_{k+2}$.

**failure detector**()
**cobegin**
  **for each** path segment $\pi \in P_r$:
    $suspect_r^\tau[]$ = { } // the set of unreliable path segments that r detects during $\tau$
    **while** (true) {
      **synchronize** with the router r' at other end of $\pi$;
      **collect** traffic information $info_r^{\pi,\tau}$ about $\pi$ for an agreed-upon interval $\tau$;
      **exchange** $[info_r^{\pi,\tau}]_r$ and $[info_{r'}^{\pi,\tau}]_{r'}$ with r' through $\pi$;
      **if** $\neg TV(\pi, info_r^{\pi,\tau}, info_{r'}^{\pi,\tau})$ **then**
        $suspect_r^\tau[] = suspect_r^\tau[] \cup \{\langle \pi \rangle\}$;
        **reliable broadcast** $([\pi]_r)$;
    }
**coend**

Figure 3: $\Pi_{k+2}$, A $(k+2)$–Complete and $(k+2)$–Accurate Protocol

# References


[1] Xuhui Ao. DIMACS Report: Workshop on Large Scale Internet Attacks, November 2003.

[2] Kirk A. Bradley, Steven Cheung, Nick Puketza, Biswanath Mukherjee, and Ronald A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 115–124, May 1998.

[3] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[4] S. Cheung. An efficient message authentication scheme for link state routing. In *ACSAC*, pages 90–98, 1997.

[5] Steven Cheung and Karl Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *New Security Paradigms Workshop*, 1997.

[6] Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3:14–30, 1982.

[7] Gauis. Things to do in Ciscoland when you're dead, January 2000. www.phrack.org.

[8] Michael T. Goodrich. Efficient and secure network routing algorithms, January 2001. Provisional patent filing. citeseer.nj.nec.com/414798.html.

[9] Kevin J. Houle, George M. Weaver, Neil Long, and Rob Thomas. Trends in denial of service attack technology. CERT Coordination Center Technical Report, October 2001.

[10] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *The 8th ACM International Conference on Mobile Computing and Networking*, September 2002.

[11] John R. Hughes, Tuomas Aura, and Matt Bishop. Using conservation of flow as a security mechanism in network protocols. In *IEEE Symposium on Security and Privacy*, pages 132–131, 2000.

[12] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure Border Gateway Protocol (Secure-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, April 2000.

[13] Craig Labovitz, Abha Ahuja, and Michael Bailey. Shining light on dark address space, November 2001. Arbor Networks Technical Report.

[14] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[15] Venkata N. Padmanabhan and Daniel R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Computer Communications Review*, 33(1):77–82, 2003.

[16] Radia Perlman. *Network Layer Protocols with Byzantine Robustness*. PhD thesis, MIT LCS TR-429, October 1988.

[17] Radia Perlman. *Interconnections: Bridges and Routers*. Addison Wesley Longman Publishing Co. Inc., 1992.

[18] Bradley R. Smith and J. Garcia-Luna-Aceves. Securing the border gateway routing protocol. In *Proc. Global Internet'96*, November 1996.

[19] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with Rocketfuel. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 133–145. ACM Press, 2002.

[20] Lakshminarayanan Subramanian, Volker Roth, Ion Stoica, Scott Shenker, and Randy Katz. Listen and Whisper: Security Mechanisms for BGP, NANOG 30, February 2004. http://www.merit.edu/ nanog/mtg-0402/subramanian.html.

[21] David Taylor. Using a compromised router to capture network traffic, July 2002. Unpublished Technical Report.

[22] Renata Teixeira, Keith Marzullo, Stefan Savage, and Geoffrey M. Voelker. In search of path diversity in ISP networks. In *Proceedings of the conference on Internet Measurement Conference*, pages 313–318. ACM Press, 2003.

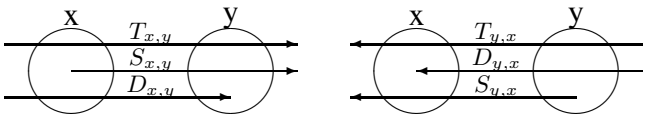
[23] Rob Thomas. ISP Security BOF, NANOG 28, June 2003. http://www.nanog.org/mtg-0306/pdf/thomas.pdf.

[24] Shyhtsun F. Wu, Fei yi Wang, Brian M. Vetter, Rance Cleaveland, Y. Frank Jou, Fengmin Gong, and Chandramouli Sargor. Intrusion detection for link-state routing protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, 1997.

# Appendix

## 7.1 WATCHERS

The WATCHERS protocol, which detects and isolates faulty routers based on a distributed network monitoring approach, was developed (and criticized) at the University of California, Davis from 1997 through 2000 [5, 2, 11].A faulty router is defined to be one that drops or misroutes packets or that behaves in an arbitrary manner with respect to the WATCHERS protocol. Cheung and Levitt [5] first proposed to use a *conservation of flow principle* (CoFP) to detect faulty routers. Basically, CoFP states that each input to a router should either be absorbed at that router or passed along to another routers.




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

Figure 4: Transit packet byte counters

As shown in Figure 4, each router counts how many bytes it has received and forwarded through each link during an agreed-upon time interval. Each router then floods the snapshots of its counters. Once it has these counters, it uses a two-phase protocol to determine which routers are faulty. The two phases are:

1. **Validation:** A router $a$ compares, for each neighbor $b$, its counters for the $a-b$ link with those of $b$. If the counters do not agree, it detects its neighbor as faulty. Similarly, for each neighbor $b$ and each of its neighbor $c$, $a$ compares the $b-c$ link counters of $b$ with those of $c$. If these counters do not agree, then $a$ knows that at least one of $b$ and $c$ is faulty, and so $a$ does nothing further with $b$; it assumes that $b$ will detect $c$ as faulty or vice versa.

2. **Conservation of flow test:** If the validation phase is passed successfully, then $a$ checks if each neighbor $b$ preserves CoFP. It does so by computing the incoming transit flow $I_b$ and the outgoing transit flow $O_b$ of router $b$:

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

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

In this scheme, each router maintains six counters for each of its neighbors.[3] Thus, if $R$ is the maximum connectivity in the network, then the space cost per router of this protocol $O(R)$. Since all counters are compared over the same time interval, all of the routers periodically synchronize with each other.

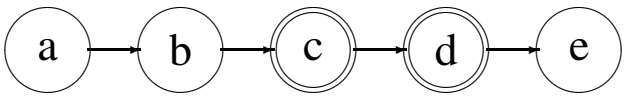



Figure 5: Consorting routers

[3] In fact, each router maintains seven counters for each of its neighbors. The seventh counts misrouted packets by that neighbor. Whenever this counter is nonzero, it is identified as a faulty router.

Later, the architects of WATCHERS noticed that this algorithm was not sufficient to detect *consorting faulty routers* [16], defined as the faulty routers launching a coordinated attack and cooperating to hide each other's malicious behavior. For example, in Figure 5, let $a$ send packets to $e$ through $b, c$, and $d$. If $c$ and $d$ are consorting faulty routers then they can drop all packets and still hide this attack by simply increasing their $D_{c,d}$ counters rather than $T_{c,d}$. With the motivation of this scenario, Bradley *et al.* extended the results in [5] and presented the final version of the WATCHERS protocol [2]. In this version, each router maintains a separate set of counters for each neighbor and final destination of each packet. In the example, when $a$ sends the packet, it updates its $S^e_{a,b}$ counter. $b$ updates its $T^e_{b,c}$ counter after forwarding the packet. $c$ and $d$ now cannot simply drop the packets and hide the attack just by updating some of their counters. In this scheme, the space required at a router is $O(RN)$, where $N$ is the total number of routers in the network.

WATCHERS' was designed assuming:

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

Two years later, another group at UC Davis (Hughes *et al.* [11]), argued that CoFP is inappropriate to use as a security mechanism in network protocols. They mentioned three general scenarios in which WATCHERS does not work:

- Ones for which WATCHERS can be fixed with small modifications in the protocol such as source routing, premature aging.
- Ones that are not addressed by WATCHERS such such as packet modification and packet fabrication. These could be addressed with a more general traffic validation mechanism.
- Ones that represent attacks on the control plane such as ghost routers, and "hot potato" examples in [11] where faulty routers announce incorrect LSPs.

Perhaps more interesting, they did not notice that WATCHERS failed to detect one case of consorting routers. Consider two faulty routers $c$ and $d$ in Figure 5. Assume that there is another (unshown) set of bidirectional links connecting $a$, $b$ and $e$ so that the *good path condition* is satisfied. Thus, all of the system requirements are met. Assume that $c$ drops packets it sends along the $c - d$ path but it does not reflect this in $T^e_{c,d}$. Router $d$ can have a correct value of $T^e_{c,d}$ that is inconsistent with $c$'s counter $T^e_{c,d}$, which means that their neighbors $b$ and $e$ will not perform conservation of flow test for $c$ or $d$ respectively. Router $d$, being faulty, need not detect $c$ as faulty.

Our concerns about WATCHERS differs from the criticisms of [11]. First, there is no specification of the problem it solves, which makes it hard to compare with other protocols. It assumes a somewhat weak threat model (routers can't be arbitrary faulty) and it has some system requirements whose importance isn't clear (namely, global synchronization and the *good neighbor condition*). It doesn't detect all faulty routers, and the amount of state each router must maintain is bounded from above only by the total number of routers in the network.

## 7.2 Basic Theorems

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

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

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

For each path segment $\pi$ of $\Pi$ that contains $r$, $r$ is t-faulty in $\pi$ during $\tau$. Given *bad*$(k)$, $r$ can be in a group of no less than one and no more than $k$ adjacent faulty routers. This group, by definition, is bounded on both sides by nonfaulty routers. ■

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

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

## 7.3 Properties of $\Pi_2$

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

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

**Theorem 4** *The protocol $\Pi_2$ is 2**–FC Complete**.*

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

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

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

Both $f$ and $\ell$ compute *TV*($\pi''$, $info_f^{\pi'',\tau}$, $info_\ell^{\pi'',\tau}$) to be false. By Theorem 2, there exists a 2-path segment $\pi = \langle i, i+1 \rangle$ such that *TV*($\pi''$, $info_i^{\pi'',\tau}$, $info_{i+1}^{\pi'',\tau}$) is false where $f \leq i < i+1 \leq \ell$. Since all routers between $f$ and $\ell$ are faulty and fault-connected to $r$, at least one of {i, i+1} is faulty and fault-connected to $r$.

Both correct routers $f$ and $\ell$ detects this failure and reliably broadcasts to all correct routers with the evidence of $info_i^{\pi'',\tau}, info_{i+1}^{\pi'',\tau}$ that are digitally signed by routers $i$ and $i+1$, respectively. Eventually all correct routers suspect $\pi = \langle i, i+1 \rangle$. ■

### 7.4 Properties of $\Pi_{k+2}$

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

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

For a correct router, to suspect a path segment $\pi$, router $s$ that is either the first or last router of $\pi$ announces that '$\pi$ is unreliable'.

1. If this announcement is incorrect, then $s$ is p-faulty.
2. If this announcement is correct, then $s$ found *TV*($\pi$, $info_1^{\pi,\tau}$, $info_{|\pi|}^{\pi,\tau}$) to be false. Assume there exists no faulty router in $\pi$ exhibiting faulty manner with respect to $\pi$ during $\tau$. Thus, each router in $\pi$ forwards the traffic traversing $\pi$ correctly. Since both router 1 and router $s$ are correct, they collection and exchange traffic information correctly. Thus, both routers will find*TV*($\pi$, $info_1^{\pi,\tau}$, $info_{|\pi|}^{\pi,\tau}$) to be true, which contradicts our assumption.

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

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

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

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

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

$f$ and $\ell$ monitor $\pi$ and apply the protocol $\Pi_{k+2}$ for $\pi$. After exchanging their traffic information, both $f$ and $\ell$ compute *TV*($\pi$, $info_f^{\pi,\tau}$, $info_\ell^{\pi,\tau}$) to be false and suspect $\pi$ and disseminate this information to the all other correct routers by reliable broadcast. Since $\pi$ contains a t-faulty router $r$ and the length of $\pi$ might be at most $k+2$, the protocol $\Pi_{k+2}$ is (k+2)–Complete. ■