Detecting and Isolating Malicious Routers

Alper Tugay Mızrak, *Student Member*, *IEEE*, Yu-Chung Cheng, Keith Marzullo, *Member*, *IEEE*, and Stefan Savage, *Member*, *IEEE*

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 are subverted to act in a malicious fashion. By manipulating, diverting, or dropping packets arriving at a compromised router, an attacker can trivially mount denial-of-service, surveillance, or man-in-the-middle attacks on end host systems. Consequently, Internet routers have become a choice target for would-be attackers and thousands have been subverted to these ends. In this 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 a concrete protocol that is likely inexpensive enough for practical implementation at scale. Finally, we present a prototype system, called Fatih, that implements this approach on a PC router and describe our experiences with it. We show that Fatih is able to detect and isolate a range of malicious router actions with acceptable overhead and complexity. We believe our work is an important step in being able to tolerate attacks on key network infrastructure components.

Index Terms—Communication/networking and information technology, network-level security and protection, network protocols, routing protocols, fault tolerance.

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, toward its ultimate destination. Therefore, if a router is compromised, it stands to reason that an attacker may drop, delay, reorder, corrupt, modify, or divert *any* of the packets passing through. 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.

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 or latent software vulnerabilities [1], [2], [3]. One network operator recently documented more than 5,000 compromised routers as well as an underground market for trading access to them [4]. 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 enough 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 [5], [6]. 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.

One approach to this problem is to detect the act of router intrusion as it happens. For example, traditional host-based intrusion detection techniques, such as system call anomaly analysis [7], [8], [9], could be applied to the router environment as well. However, this approach also has the same limitations as in the host environment—once a router is compromised, the detection software is no longer dependable. Indeed, it has become increasingly common for malware to disable antivirus and IDS products and we see no reason to believe the router environment will be different. Thus, while such attack detection mechanisms are undoubtedly an important defensive element, in this paper, we start from the assumption that routers may be successfully compromised.

Given this threat model, the problem of detecting a compromised router falls to its neighbors: A compromised router can potentially be identified by correct routers when it deviates from exhibiting expected behavior. This overall approach can be broken into three distinct subproblems:

1. Traffic validation. Traffic information is the basis of detecting anomalous behavior: Given traffic entering a part of the network, and an expected behavior for

• *The authors are with the Computer Science and Engineering Department, University of California at San Diego, EBU3B, 9500 Gilman Drive, La Jolla, CA 92093-0404.*
E-mail: {amizrak, ycheng, marzullo, savage}@cs.ucsd.edu.

Manuscript received 14 Sept. 2005; revised 12 Apr. 2006; accepted 25 Apr. 2006; published online 3 Aug. 2006.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0123-0905.

the routers in the network (i.e., a known routing configuration), anomalous behavior is detected when the monitored traffic leaving one part of the network differs significantly from what is expected. However, implementing such validation practically can be quite tricky and requires tradeoffs between the overhead of monitoring, communication, and accuracy.

2. Distributed detection. It is impossible for a single router to establish that its neighbor is anomalous. Thus, detection requires synchronizing a collection of traffic information and distributing the results so anomalous behavior can be detected by *sets* of correct routers.
3. Response. Once a router, or set of routers, is thought to be faulty, the forwarding tables of correct routers must be changed to avoid using those compromised nodes. In addition, over longer time scales, an appropriate alert must be raised so human forensic experts can respond appropriately.

This paper addresses each of these problems in turn. For different threats, we describe a range of appropriate and efficient traffic validation functions. Next, we examine how these functions can be used to build an anomalous behavior detector for compromised routers. Finally, we show how this detector can be integrated into link-state routing to automatically isolate network paths demonstrating anomalous behavior. This paper is substantially extended version of [10].

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 is often referred to as an attack on the *control plane*, while the second is termed 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 [11], 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 based on ensuring the authenticity of route updates and detecting inconsistency between route updates [12], [13], [14], [15], [16], [17], [18], [19].

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 [11], [20]. Perlman developed a novel method for robust routing based on source routing, digitally signed *route-setup packets*, reserved buffers. However, many implementation details are left open and the protocol requires higher network level participation to detect anomalies. 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 [12] does this by comparing TCP Data and Acknowledgment packets to provide evidence that a path is part of end-to-end connectivity. This approach only tests for gross connectivity and cannot reveal whether packets have been diverted, modified, created, reordered, or selectively dropped. Padmanabhan and Simon's Secure Traceroute [21] achieves a similar goal monitoring the traffic to the intermediate routers. Recently, Avramopoulos et al. [22], [23] present a secure router routing a combination of source routing, hop by hop authentication, end-to-end reliability mechanisms, and timeouts. But, it still has a high overhead to be deployable in modern networks. The approach most similar to our own is the WATCHERS [24], [25] protocol, which detects disruptive routers based on a distributed network monitoring approach and a traffic invariant called conservation of flow. 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. [26]. 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). Herzberg and Kutten [27] presents an abstract model for Byzantine detection of compromised routers based on timeouts and acknowledgments from the destination and possibly from some of the intermediate routers to the source. The requirement of information from intermediate routers offers a trade-off between fault detection time and message communication overhead. This trade-off is analyzed within the given abstract model.

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 as consisting of individual routers interconnected via directional point-to-point links. In using this model, we purposely ignore several real-life complexities. For example, real routers are not homogeneous nodes, but in fact represent a collection of independent network

interfaces which are themselves interconnected and are, in practice, addressed and controlled distinctly. We have eliminated this detail for the convenience of description; our analysis can easily accommodate this expanded model. Similarly, we have chosen to ignore the possibility of broadcast channels in our model since they are rare in today's wired networks and can be easily incorporated as collections of point-to-point links (although there may be opportunities for additional optimization in broadcast environments).

We assume that each router has sufficient data processing capability to generate traffic summaries describing the network traffic they are forwarding. For the most precise summary functions this may involve touching all packet content. This is well within the capabilities of modern ASICs, but might require sampling in software implementations. We also assume that each router has sufficient computational capability to exchange and reconcile summaries with its neighbors. We discuss the issue of overhead further in Section 4 and Section 9, but, in general, our algorithms are efficient and practical for existing router CPUs.

Within a network, we presume that packets are forwarded in a hop-by-hop fashion—each router following the directions of a local forwarding table. As well, we assume that these forwarding tables are updated via a distributed link-state routing protocol such as OSPF or IS-IS. This is critical, as we depend on the routing protocol to provide each node with a global view of the current network topology, which we assume to be *consistent*. Indeed, this assumption is valid in practice since studies of OSPF behavior have shown that in-network topology changes are extremely rare [28], [29]. When topology changes do occur there may be transient inconsistencies between routers that make it impossible to determine whether traffic is being forwarded correctly or not. Thus, if a compromised router frequently changes its connectivity, it may hide some forwarding attacks. For this reason, among others, any secure network infrastructure *also* requires software to detect security violations or anomalies in the control plane [12], [13], [14], [15], [16], [17], [18], [19].

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

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

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, these are called *terminal routers*. A path might consist of only one router, in which case the source and sink are the same.

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

We do not rely on source routing, as has been done by some work in the past [11], [22], [27]. We do assume some knowledge of the path a packet will take, at least in the stable state. In link state protocols, this can be problematic because they can take advantage of multiple paths with equal cost for load balancing purposes. Fortunately, the current generation of routers uses a deterministic hash algorithm to spread the traffic load across available interfaces (e.g., Cisco Express Forwarding [30] and Juniper routers with an Internet Processor ASIC [31]). Thus, a router can predict the path a packet will take in the stable state based on its own routing tables and the hash functions.

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. [32] 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 hosts on local-area networks—single workstations rarely have multiple paths to their network infrastructure. In these situations, for fate-sharing reasons, there is little that can be done. If the host's access router is compromised, then the host is partitioned and there is no routing remedy even if an anomaly is detected; the fate of individual hosts and their access routers are directly intertwined. Moreover, from the standpoint of the network such traffic *originates* from a compromised router, and, therefore, cannot demonstrate anomalous forwarding behavior.[1] To summarize, our protocols are designed to detect anomalies between pairs

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

of *correct* nodes. Thus, for simplicity, we assume that a terminal router is not faulty with respect to traffic originating or being consumed by that router.

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

4 Traffic Validation

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

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

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

- π is a path segment $\langle r_1, r_2, \ldots r_x \rangle$ whose traffic is to be validated between routers r_i and $r_j \in \pi$.
- $info(r, \pi, \tau)$ is information about traffic that router r forwarded to π over some time interval τ.
- If routers r_i and r_j are not faulty, then

$$TV(\pi, info(r, \pi, \tau), info(r_j, \pi, \tau))$$

  evaluates to *false* iff π contains a router r that was faulty in forwarding traffic along π during τ, and router r is between routers r_i and r_j within π.

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

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

Note that we have already made one engineering decision: We describe aggregate traffic rather than individual packets. This is in contrast to some prior works, e.g., Perlman [11], Herzberg et al. [27], and Avramopoulos et al. [22]. While we compute state over each packet locally, limiting distributed reconciliation to traffic aggregates amortizes the communication and synchronization overhead (otherwise, prohibitive) across many packets. This also makes it feasible to apply a threshold mechanism to distinguish between acceptable bad behavior (e.g., small amounts of packet loss and reordering) and malicious behavior.

4.1 Characterizing Traffic

While the most precise description of traffic sent into the network is an exact copy of that traffic, many characteristics of the traffic can be summarized far more concisely. In particular, if we concentrate on particular threats—ways in which a malicious entity might alter the traffic—we can limit our effort to detecting just those actions.

One limitation of previous work has been that they assumed limited kinds of threats. For example, `WATCHERS` [26] and Herzberg et al. [27] effectively detect only packet losses that result from a compromised router. In all of the previous work, reordering attacks have been ignored, even though reordering of packets can effect TCP performance tremendously [33], [34]. It would not be hard to extend Secure Traceroute [21] to include this kind of attack, but it cannot be done with the protocols of Perlman [11], Herzberg et al. [27] and Avramopoulos et al. [22] since they monitor a single packet at each round. We instead divide up arbitrary behavior into five different threats:

1. *Packet loss*. A compromised router can drop any subset of the packets. As per Almes et al. [35], loss can be measured as the amount of data arriving at the sink of path segment subtracted from the amount of data sent from its source.
2. *Packet fabrication*. A compromised router can generate packets and inject them into the traffic stream. This can be measured as the number packets which are reported at the sink of a packet segment but not monitored as being sent by its source.

   Misrouting packets can be considered an instance of both packet loss and packet fabrication.
3. *Packet modification*. One can consider this threat as a combination of packet loss and fabrication, but it may not be detectable by simply comparing the number of packets arriving at the sink with the number sent from the source. Instead, some summary of the content needs to be maintained, and one measures the number of modified packets.

4. *Packet reordering*. A compromised router can reorder packets. Doing this can lead to performance problems or, in the extreme, denial of service. There are many reasonable and incompatible methods of measuring the amount of reordering, e.g., [33], [34], [36], [37]. We use the definition from [36] because of its simplicity: Given a transmitted stream S and a received stream F, remove from both all lost, fabricated, and modified packets. Then, find the longest common subsequence ℓ between these modified S and F. The amount of reordering is defined as $|S| - |\ell|$.
5. *Time behavior*. A compromised router can delay traffic. Like reordering, doing this can lead to performance problems or, in the extreme, denial of service. There are simple metrics one can use, such as the first n moments of the interpacket delay distribution. However, such metrics are notoriously sensitive in packet networks; we believe this area requires more research.

These threats completely cover the set of bad behaviors a router can exhibit in forwarding data. When all of these metrics are zero, then no router is forwarding traffic in a faulty manner.

4.2 Traffic Summary Functions

We have implemented three traffic summary functions, each reflecting a different trade off among accuracy, completeness, and threat. They are:

- *Conservation of Flow*. Our first summary function resembles the "conservation of flow" approach used by the WATCHERS protocol [25]. The traffic information $info(r_i, \pi, \tau)$ collected by router r_i consists of two counters, each counting the number of packets in one of the two directions of traffic along the path π. Periodically, r_i passes these counters to other routers collecting information about π. Traffic validation is done by comparing the values of these counters. In general, this is a fragile summary function because it only detects actions that cause packet losses, and it assumes that malicious routers cannot fabricate packets to "fudge" the counts appropriately. However, it is extremely cheap to implement, both in the per packet cost and in the associated overhead to communicate the traffic information among routers. It might be possible to extract such information from existing traffic analysis tools, such as Cisco's Netflow [38], without requiring router modifications although we have not attempted this (there are a number of complexities in how Netflow manages flow records that pose nontrivial synchronization challenges).
- *Conservation of content*. To detect modification of packets, we can use a fingerprint (that is, a hash), of the payload in place of a simple counter. Analogous to conservation of flow, each router then periodically communicates a set of packet fingerprints with other routers for traffic validation, which is calculated via set difference. In addition to detecting packet modification, this approach also detects packet loss, packet fabrication, and misrouting.

  One technical difficulty with this approach is that packets are naturally modified as they traverse routers. In particular, the TTL field in the IP header is decremented and the IP header checksum is updated. We address this by having each router compute fingerprints based on the TTL value and checksum at one end of the path π.

  One downside to this approach is that it requires storing and communicating a fingerprint for each packet forwarded by a router. This is a significant overhead. One can save significant space and bandwidth by using more sophisticated algorithms for calculating set differences. The simplest approach is to simply use Bloom filters [39] to represent the set of fingerprints. One can then use the population of the bitwise difference between the filters to calculate the size of the set difference. This approach is far cheaper to implement, but comes at some expense in accuracy. More problematic is that it is difficult to know in advance the appropriate parameters for the Bloom filter. A too-small filter can result in significant errors in estimation. A more promising approach is to leverage distributed set reconciliation algorithms [40]. This approach has greater computation overhead than Bloom filters, but it is optimal in bandwidth utilization [40].
- *Conservation of content order*. Packet reordering can be detected by maintaining ordered lists of packet fingerprints rather than simply sets. Like with conservation of content, this can result in a significant storage overhead. Traffic validation can then be done using the metric defined in Section 4.1. This has a higher overhead than simply computing the size of the set difference. We have not yet looked into methods that would reduce the storage or computational overheads.

4.3 Detection Accuracy

Any traffic validation mechanism presents the potential for false negatives—attacks which are not detected—and false positives—benign traffic that is incorrectly identified as invalid. While the strengths and weaknesses of specific mechanisms are beyond the scope of this paper (our focus here is on the general problems posed by distributed detection and the remainder of the paper considers TV to be a black box), the overall issues bear some discussion. The issue of false negatives is ultimately tied to the precision and completeness of the validation technique used. A system based on conservation of flow could not expect to detect attacks that modify packet content without changing their number. Similarly, false positives relate to how well the validation mechanism can model benign network behavior. For example, the TV mechanism must account for IP packets whose TTL field is set to 1 and will be correctly dropped at the next router. Similarly, there are some cases where traffic is legitimately tampered by the routers such as lawful intercept [41] and GRE tunneling [42]. In these cases, a traffic validation mechanism would

also need to model the associated overlay topology and classification rules.

Ultimately, there is little data about the prevalence of different router attacks and how different validation mechanisms would interact with these. Instead, we have attempted to describe attacks and associated validation mechanisms in a generic fashion. The detailed design of traffic validation mechanisms and their empirical evaluation remains an ongoing research activity.

5 Distributed Detection

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

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

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

5.1 Specification

Due to the nature of the problem, our specification is more complex than the typical accuracy and completeness properties of an imperfect failure detector [43]. Since this detector is based on evaluating traffic collected over a period of time, we have the failure detector report pairs (r, τ), which means that r was suspected as being faulty during the time interval τ. A perfect failure detector would implement the following two properties:

1. *Accuracy (tentative)*: A failure detector is *accurate* if, whenever a correct router suspects (r, τ), then r was faulty during τ.
2. *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, τ) for some τ containing t.

As noted above, though, our failure detector is based on traffic information collected by untrustworthy routers. This results in detections that are imprecise: we may not be able to pin down which router is compromised. So, we have the failure detector return a pair (π, τ), where π is a path segment. This also allows us to give more information: We restrict the detection to a router being faulty with respect to forwarding traffic along π:

- a-*Accuracy*: A failure detector is *a-Accurate* if, whenever a correct router suspects (π, τ), then $|\pi| \leq a$ and some router $r \in \pi$ was faulty in π during τ.
- 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 (π, τ) for some path segment $\pi : |\pi| \leq a$ such that r was faulty in π at t, and for some interval τ containing t.

Note that a faulty router can report an incorrect value for the traffic that traversed a path as well as alter this traffic. We use the term *t-faulty* (that is, *traffic* faulty) to indicate a router that alters traffic and the term *p-faulty* (that is, *protocol* 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 π" to indicate that the faulty behavior is with respect to traffic that transits the path π. 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 behavior 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 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 (π, τ) for some path segment $\pi : |\pi| \leq a$ such that r was t-faulty in π at t, and for some interval τ containing t.

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

Making a $bad(k)$ assumption has an effect on failure detection. Allowing compromised routers to be adjacent complicate detection further, since they can cooperate to hide the evidence that a router is faulty. For example, 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 τ, 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 the counters indicate that traffic has disappeared between them. 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 this kind of scenario 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, then they would be suspected later. We formalize this approach by defining a condition we call being *fault connected*. Given a path segment π and an interval τ, we say that a router r is fault connected to router s with respect to π if both r and s are in π, and all of the routers between s and r are faulty in π during τ. 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 (π, τ) for some $\pi : |\pi| \leq a$ and some τ containing t such that there is a router r' that was faulty in π at time t' in τ and is fault-connected to r.

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

Since we are assuming arbitrarily faulty routers, we have to allow faulty router to suspect correct routers. We address this issue in the countermeasure protocol: as with WATCHERS, the only suspicion that elicits a countermeasure 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 (as a result of faulty suspicions) adds no further disadvantage.

Using this terminology, Perlman [11] protocol is M-Accurate and M-Complete, where M is the maximum length of a path between any pair of routers. All other prior protocols we know of are 2-Accurate. However, the detection is attained at only the source router in Perlman [11], Avramopoulos et al. [22], and Secure Traceroute [21]. If other routers learn of the fault from the detecting router r, then such a router has to consider the possibility that r is p-faulty. WATCHERS [26] is neither Complete nor FC Complete, due to a flaw, which we have shown in [44]. The claims in [21] indicate that Secure Traceroute is 2-Accurate and 2-FC Complete. But, the description of the protocol (especially on how the overhead is reduced by pretending to monitor all prefixes of a path segment from a source to a destination) is not detailed enough for us to be confident of the actual accuracy and completeness of the protocol. Herzberg et al. [27] and Avramopoulos et al. [22] protocols are 2-FC Complete.[2]

5.2 $\prod_2$: A 2-FC Complete and 2-Accurate Protocol

Given a complete and accurate traffic validation mechanism, an obvious approach is to have each router collect traffic information over some agreed-upon interval, and then use consensus 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 τ for the path segment π. If at least one of the other routers is t-faulty with respect to π 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 about. 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 $k + 2$. Note that given source and destination may not be separated by at least k routers, and, so, a router also collects information about all paths of which it is a member whose length is less than $k + 2$.

The resulting protocol $\prod_2$ is shown in Fig. 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 π and collects information for the same traffic passing through π for an agreed-upon interval τ. Periodically, r sends that traffic information to all routers in π 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 π. 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 π 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

2. However, the detection has been done only at the source router. Not every router in the network agree on the detection as we specified above.

failure detector()
cobegin
  for each path segment $\pi \in P_r$:
    $suspect_r^\tau[\pi]$ = { } // the set of suspicious path segments in π that r detects during τ
    while (true) {
      synchronize with all routers in π;
      collect traffic information *info*(r, π, τ) about π for an agreed-upon interval τ;
      consensus ($[$*info*$(1, \pi, \tau)]_1, [$*info*$(2, \pi, \tau)]_2, ..., [$*info*$(|\pi|, \pi, \tau)]_{|\pi|}$);
      // at this point all correct routers in π agree on the values of *info*(i, π, τ)
      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

Fig. 1. $\prod_2$: A 2-FC Complete and 2-Accurate protocol.

transited during τ). In either case, a correct router r in π will put the 2-path segment $\langle i, i+1\rangle$ into $suspect_r^\pi[\pi]$ set, and reliably broadcast the evidence of the failure detection: $([info(i, \pi, \tau)]_i,\ [info(i+1, \pi, \tau)]_{i+1})$. Upon receiving this information, all other correct routers will evaluate $TV(\pi, info(i, \pi, \tau), info(i+1, \pi, \tau))$ as *false* and detect the fault on the 2-path segment $\langle i, i+1\rangle$.

$\prod_2$ is given in Fig. 1. In Appendix 1.2, we show that $\prod_2$ is **2-Accurate** and **2-FC Complete**.

5.3 $\prod_{k+2}$: A $(k+2)$-Complete and $(k+2)$-Accurate Protocol

$\prod_2$ 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 $info(r_1, \pi, \tau), info(r_4, \pi, \tau)$ be the traffic information that router r_1 and r_4 collect during τ. If at least one of the other routers is t-faulty with respect to π during this interval, then $TV(\pi, info(r_1, \pi, \tau), info(r_4, \pi, \tau))$ will be false. In this case, r_1 suspects $\langle r_2, r_3, r_4\rangle$. Similarly, r_4 suspects $\langle r_1, r_2, r_3\rangle$.

For this protocol, a router need not monitor as many path segments as with $\prod_2$. Instead, a 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. A trivial reason this is so is that not all path segments need be $k+2$ long. A more substantial reason is that compromised routers may hide another router's bad behavior. For example, given that $bad(2)$ 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 π, but r_4 can hide the fact that r_2 is t-faulty by simply sending traffic information to r_1 such that $TV(\pi, info(r_1, \pi, \tau), info(r_4, \pi, \tau))$ holds. If r_1 were to instead also monitor the path $\langle r_1, r_2, r_3\rangle$, then r_1 could detect r_2's faulty behavior. So, it is necessary for a router r to monitor all x-path segments for $3 \leq x \leq k+2$ of which r is an end.

For each path segment $\pi \in P_r$, r synchronizes with the other end router of π and collects information for the traffic passing through π during an agreed-upon interval τ. Router r then exchanges digitally signed traffic information with the router r' on the other end. If the exchange operation fails within a prespecified timeout interval μ, or if r finds $TV(\pi, info(r, \pi, \tau), info(r', \pi, \tau))$ is false, then there is at least one faulty router in π during τ. In particular, either r' is p-faulty or some router in π is t-faulty. So, r detects $\pi - \langle r\rangle$. However, when it announces this detection to the other routers, a correct router receiving this information suspects π since r might be faulty. For simplicity, we also have router r suspect π.

$\prod_{k+2}$ is given in Fig. 2. In Appendix 1.3, we show that $\prod_{k+2}$ is **(k+2)-Accurate** and **(k+2)-Complete**.

6 Response

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

Suppose that some path segment π is detected. An obvious countermeasure would be to remove all of the

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

Fig. 2. $\prod_{k+2}$: A $(k+2)$-Complete and $(k+2)$-Accurate Protocol.

routers in π from the routing fabric. By doing so, we avoid using any router that has been suspected of being compromised. Doing this, though, could also have a serious, and perhaps unnecessarily high, impact on network performance. A less aggressive countermeasure would be to only remove the path segment π from the routing fabric. In doing so, we may allow compromised routers to keep forwarding packets, but only along paths over which no faulty behavior has been observed.

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

7 System Architecture

We have implemented a prototype system, called Fatih, that incorporates our approach into a Linux 2.4-based router platform running OSPF. We have implemented a variety of traffic validation mechanisms mentioned in Section 4, including conservation of flow, content, and content order, as well as the $\prod_{k+2}$ distributed detection algorithm explained in Section 5.3. The system architecture, shown in Fig. 3, consists of five principal components.

7.1 FATIH Coordinator

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

1. Based on the network topology exported by the Routing Daemon, the Coordinator decides which path segments to monitor depending on the system parameter k (Section 5). We have configured our prototype with $k = 1$, thus each router monitors all 3-path segments originating from itself. Beside simplicity, our implementation focuses on this point of the design space because it reflects the most common capabilities available to an attacker. For an adversary to compromise adjacent routers in a manner that they attempt to conceal their actions (i.e., $k > 1$) is considerably more difficult as this requires an adversary to modify the executable protocol code running on the routers.

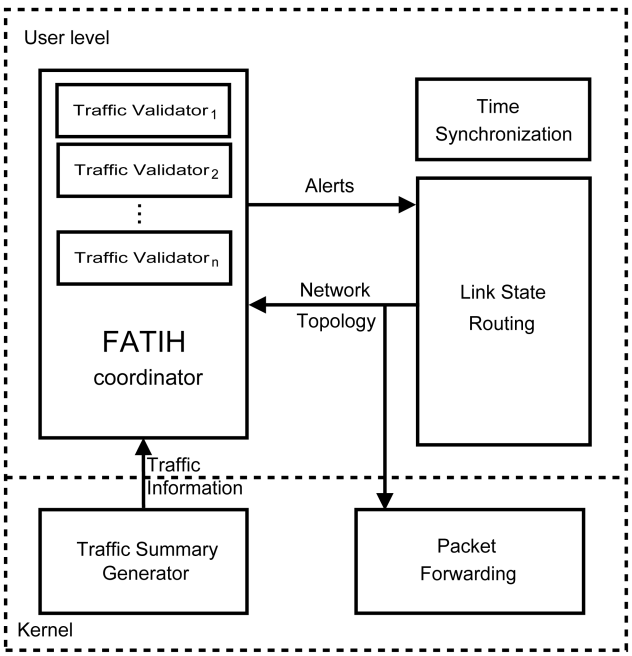


Fig. 3. Fatih system architecture.

2. The Coordinator receives information collected by the Traffic Summary Generator, determines the path being traversed, and delivers the received information to the corresponding Traffic Validator modules.
3. Finally, the Coordinator, schedules and synchronizes validation rounds among the Traffic Validators. In the current implementation, rounds are configured at a five second interval. A longer time interval requires more traffic summary state to be maintained, while a shorter time interval places more stringent synchronization requirements on the system.

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

7.2 Traffic Summary Generator

As described in Section 4, the Traffic Summary Generator updates traffic information with each forwarded packet. Depending on the underlying validation mechanism, the generator computes a packet fingerprint using the *UHASH* function [45] and associates a timestamp with it. The performance of this module is critical and, therefore, we have implemented it in the kernel to avoid unnecessary copies of packet contents. Traffic summaries are ultimately passed back to the Coordinator.

7.3 Link-State Routing Daemon

As described in Section 3, Fatih cooperates with a standard link state routing protocol. The Routing Daemon, which is based on Zebra [46] in the current prototype, implements the OSPF [47] protocol and manages link state announcements, shortest path computation and forwarding table calculation and installation. In addition, we have modified the protocol to incorporate input from Fatih. When the Routing Daemon receives an alert from the Coordinator (or via link-state updates from other routers), it recomputes new shortest path routes to avoid any suspicious path segments identified. This alert is then flooded via the link-state update mechanism to allow other routers to exclude the suspicious path segments as well.

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

7.4 Packet Forwarding

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

7.5 Time Synchronization

Fatih requires routers to have clocks synchronized closely enough so that there is not a significant disagreement over the intervals that traffic information is collected. For our purposes, clocks that are synchronized within a few milliseconds is sufficient. We use NTP [48] to synchronize the routers clocks.

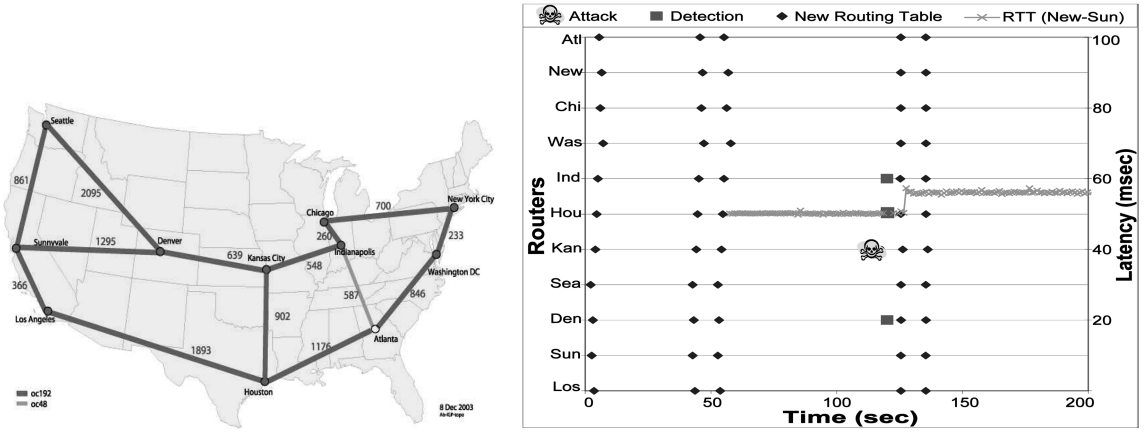
8 Experiences

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

We have chosen a topology based on the Abilene network [49], Fig. 4a, because its structure, link delays, bandwidths, and link-state metrics are all public. As well, the topology has sufficient connectivity to demonstrate the dynamics of Fatih during an attack.

We represent each Abilene Point of Presence (POP) as a single router and configure the link delay and routing metrics accordingly. Each of these routers is in turn emulated by a User-Mode Linux [50] virtual machine, configured with 64MB of memory and implementing the Fatih system architecture as described earlier. The routers are interconnected through Ethernet bridging in the Linux host operating system, modified to emulate configured link delays. The host system is a 2.6Ghz Pentium 4 server with 1GB of physical memory. Although our emulation testbed is incapable of processing the traffic volume of a real Internet backbone, it handles sufficient traffic to demonstrate Fatih's key behaviors.

Fig. 4b demonstrates Fatih in progress. Time is shown on the x-axis in seconds, different routers are shown on the left y-axis, and the right y-axis is used for the latency measured between the $New\ York$ and $Sunnyvale$ during the experiment. At the beginning of the experiment, each router discovers its immediate neighbors, transmits and receives OSPF link state updates, and computes new routing tables with the most recent link state database. After roughly 55 seconds, all routers have agreed on a common network topology and a stable forwarding path is available between all pairs of routers.



(a) (b)

Fig. 4. (a) Abilene network topology. (b) Fatih in progress.

At this point in time, we inject a synthetic traffic load into the network and initiate round trip time (RTT) measurements between $New\,York$ and $Sunnyvale$. Initially, the forwarding path between these routers is

$$\langle Sunnyvale, Denver, Kansas\,City, Indianapolis, Chicago, New\,York\rangle$$

with a configured one-way latency of 25 ms in the configuration files. As expected the measured round-trip time is roughly 50 ms.

At roughly 117 seconds, our simulated attacker compromises the $Kansas\,City$ router and modifies its behavior such that 20 percent of its transit traffic is dropped or altered. We chose the $Kansas\,City$ router as a victim because most intercoastal traffic traverses it and is therefore a obvious target. Using the Fatih protocol, the path segments

$$\langle Denver, Kansas\,City, Indianapolis\rangle,$$
$$\langle Denver, Kansas\,City, Houston\rangle, \text{ and}$$
$$\langle Houston, Kansas\,City, Indianapolis\rangle$$

are all validated every $\tau = 5$ seconds by their terminal routers. Thus, at the end of the current traffic validation round (about 3 seconds after the attack), $Denver$, $Houston$, and $Indianapolis$ detect that traffic through these monitored path segments is inconsistent and notify their OSPF routing daemons.

There are two parameters of the OSPF routing protocol that affect the following events. OSPF_delay_time is the time passed before computing a new routing table as a result of a triggering event (e.g., a new link-state update message or an alert, as in this case). $OSPF\_hold\_time$ is the time passed before any consecutive routing table computations. These values default to 5 seconds and 10 seconds, respectively, in the Zebra OSPF implementation. As a result, an additional 15 seconds pass before the detected traffic inconsistency causes the associated path segments to be removed from the routing topology. At roughly 135 seconds, this process completes and the path between New York and Sunnyvale is changed to

$$\langle Sunnyvale, Los\,Angeles, Houston, Atlanta, Washington\,DC, New\,York\rangle.$$

This new path has a configured one-way latency of 28 ms, thus the measured RTT becomes 56 ms. Note that the $Kansas\,City$ router continues to operate, but its neighboring routers will no longer forward traffic through it.

9 Analysis of the Protocols

In this section, we consider the overhead of our approach.

9.1 Fingerprinting

Like any protocol that detects packet alteration [11], [21], [22], our *Conservation of content* and *Conservation of content order* traffic summary functions compute a fingerprint for each packet. To be practical, computing a fingerprint must have a low overhead. One possibility for fingerprinting is CRC32. This is already computed by most network interfaces at line rate and is a good uniform hash function. However, it is reversible and an adversary could create modified packets that have the same hash. In some circumstances, such as a compute limited adversary, it might serve as a good fingerprinting function.

We implemented fingerprinting using UHASH which is an unkeyed version of the UMAC algorithm [45]. This algorithm allows a trade off between security and performance and is designed to support parallel implementations. Black et al. [45] and Rogaway [51] demonstrated UHASH performance of more than 1Gbps on a 700Mhz Pentium III processor computing a 4 byte hash value and delivering a 2^{-30} forging probability (which is more than sufficient for our application). In hardware, of course this performance could be increased still further.

Between software and hardware are emerging network processors which are designed to perform highly parallel actions on data packets [52]. Recently, Feghali et al. [53] described an implementation of DES [54] and AES [55][3] on the Intel IXP28xx network processors that was able to keep

3. DES and AES are two well-known encryption algorithms that provide confidentiality.

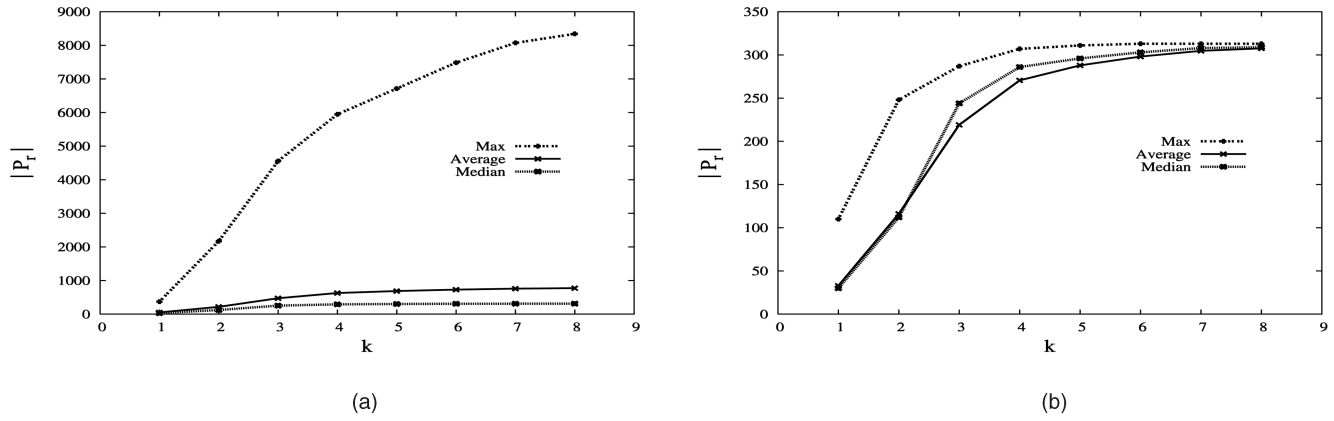


Fig. 5. Based on $bad(k)$; maximum, average, and median size of P_r that a router monitors in (a) $\prod_2$ and (b) $\prod_{k+2}$.

pace with a 10Gigabit/sec forwarding rate. Since DES and AES encryption algorithms are significantly more expensive to compute than functions such as UHASH, we can infer that it is possible to compute these packet fingerprints up to 10Gbps.

However, if there are insufficient computational resources, one can easily trade off accuracy for overhead by subsampling the packets to be considered. As in Duffield and Grossglauser's Trajectory Sampling [56], if the same random hash function is used to subsample packets at each end of a path segment, then each router should observe the same subset of packets. Further, each pair of routers is free to select such sampling functions independently and need not rely on a global secret.

9.2 State Size

One of the most important factors in terms of a protocol being practical is the size of the state at each router that needs to be maintained. Assume that we wish to provide fault-tolerant forwarding between each source and destination pair in the network. Let N be the number of routers in the network, R be the maximum number of links incident on a router, and k is the value used in the $bad(k)$ assumption. Perlman [11], Herzberg et al. [27], and Avramopoulos et al. [22] require $O(N^2)$ state, since each individual router maintains state for each (source, destination) pair. WATCHERS [26] reduces the state requirement to $O(RN)$, where each individual router keeps seven counters for each its neighbors for each destination. In Secure Traceroute [21], if a router monitors paths to all destinations, then $O(N)$ space is required (assuming that there is only one path in active use between a given source and destination).

Our protocols require a router r to record state for each path segment P_r that it monitors. By construction, this is $O(k \times R^{k+1})$ for $\prod_{k+2}$ and $O(\min\{R^{k+1}, N\})$ for $\prod_{k+2}$. In practice, though, we expect $|P_r|$ to be much smaller. For example, we have examined two network topologies, Sprintlink and EBONE, that were measured by the Rocketfuel project [57]. We counted the number of distinct path segments that a router monitors for different values of k in $bad(k)$ assumption. For instance, 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 Fig. 5, the maximum, average, and median of $|P_r|$ that a router monitors is given for this network for $\prod_2$ and $\prod_{k+2}$. For this network (as well as for EBONE), the empirical results are much smaller than the theoretical upper bounds.

As a point of comparison, for the Sprintlink network, on average, a router under WATCHERS would maintain approximately 13,600 counters, and the maximum number of counters that a router maintains is 99,205. With our conservation of flow traffic summary function[4] for $\prod_{k+2}$, a router maintains 232 counters on average and 496 in the worst case if $bad(2)$ holds. Even with the weak assumption of $bad(7)$, each router maintains 616 counters on average and 626 in the worst case.

9.3 Synchronization

Protocols that validate aggregate traffic requires synchronization to agree on a time interval to collect traffic information. WATCHERS synchronizes all routers using a snapshot algorithm [58]. In $\prod_2$, for each path segment π in P_r, a router r synchronizes with all the routers in π to agree on when and for how long the next measurement interval τ will be. 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. Perfect synchronization would not be necessary in practice, since the traffic validation function TV could be written to accommodate a small skew. The synchronization requirements for $\prod_{k+2}$ are lower than for $\prod_2$. As for each path segment π that a router r monitors, r needs to agree with only the other end router r' of π. This is the same requirement for Secure Traceroute.

9.4 Information Dissemination

$\prod_2$ requires each router in path segment π to reach consensus about the traffic information collected over π during a time interval τ. To do so necessitates 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 [59]. For $\prod_{k+2}$, in order to exchange traffic information, neither Consensus nor the *good neighbor*

4. This requires two counters per path segment, one for each direction.

condition of WATCHERS is required (*good neighbor* requires that each compromised router is adjacent to a noncompromised router). Furthermore, with $\prod_{k+2}$, 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. To avoid impersonation attacks, $\prod_{k+2}$ requires authentication. Meanwhile, the final reliable broadcast of both $\prod_2$ and $\prod_{k+2}$ can be done as part of the LSA distribution of the link state protocol.

9.5 Multicast

Multicast forwards traffic along more than one path. Since conservation of flow as stated by WATCHERS inherently assumes single-path communication, WATCHERS cannot be easily extended for multicast communications. All of the other protocols discussed above, including our own, can be extended easily for multicast. Doing so would require the routers to agree on the network topology and the multicast tree (something that, for example, with MOSPF [51], would be straightforward to do).

9.6 Fragmentation

Fragmentation does not change the data that is carried in network traffic, but rather changes the way that data is packetized. Thus, traffic validation based on packets is sensitive to fragmentation, while traffic validation based on data is not. Of the protocols mentioned in this paper, only WATCHERS bases its traffic validation just on the data (it uses byte count) and is therefore insensitive to fragmentation. However, fragmentation occurs very rarely in the Internet [52], and, so, we do not consider sensitivity to fragmentation to be a practical concern.

10 Conclusion

In this paper, we motivated and specified the problem of detecting routers with incorrect packet forwarding behavior and provided a general framework for formally analyzing protocols addressing this problem. Our approach is to separate the problem into three subproblems: 1) determining the traffic information to record upon which to base the detection, 2) synchronizing routers to collect traffic information and distributing this information among them so detection can occur, and 3) taking countermeasures when detection occurs. We presented a concrete protocol, $\prod_{k+2}$, that is likely inexpensive enough for a practical implementation at scale. We believe our work is an important step in being able to tolerate attacks on key network infrastructure components.

Appendix

Properties of the Protocols

A.1 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 π, such that:*

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

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

For each path segment π of Π that contains r, r is t-faulty in π during τ. 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 π,*

$$TV(\pi, info(h, \pi, \tau), info(j, \pi, \tau))$$

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. □

A.2 Properties of $\prod_2$

Theorem 3. *The protocol $\prod_2$ 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 π 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 $\prod_2$ 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 τ containing t such that there is a router r' that was faulty in π at time t' in τ 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 π', such that: $r \in \pi'$; r is also t-faulty in π' during τ containing t; only the first and last routers of π' (which we'll call f and ℓ) are correct and $3 \leq |\pi'| \leq k+2$.

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

Both f and ℓ 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 ℓ are faulty and fault-connected to r, at least one of $\{\mathrm{i}, \mathrm{i}+1\}$ is faulty and fault-connected to r.

Both correct routers f and ℓ 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 $\pi = \langle i, i+1 \rangle$. □

A.3 Properties of $\prod_{k+2}$

Theorem 5. *The protocol $\prod_{k+2}$ 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 π during τ.

For a correct router, to suspect a path segment π, router s that is either the first or last router of π announces that "π 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 π exhibiting faulty manner with respect to π during τ. Thus, each router in π forwards the traffic traversing π 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 $\prod_{k+2}$ to x-path segments where $x \leq k+2$. Hence, $\prod_{k+2}$ is (k+2)-Accurate. □

Theorem 6. *The protocol $\prod_{k+2}$ 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 π at t, and for some interval τ containing t.

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

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

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

ACKNOWLEDGMENTS

This work was supported by US National Science Foundation Grant 0311690.

REFERENCES

[1] X. Ao, "Report on DIMACS Workshop on Large-Scale Internet Attacks," Sept. 2003, http://dimacs.rutgers.edu/Workshops/Attacks/internet-attack-9-03.pdf.

[2] K.J. Houle, G.M. Weaver, N. Long, and R. Thomas, "Trends in Denial of Service Attack Technology," CERT Coordination Center, technical report, Oct. 2001, http://www.cert.org/archive/pdf/DoS_trends.pdf.

[3] C. Labovitz, A. Ahuja, and M. Bailey, "Shining Light on Dark Address Space," Arbor Networks, technical report, Nov. 2001, http://research.arbornetworks.com/downloads/research38/dark_address_spa%ce.pdf.

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

[5] ? Gauis, "Things to Do in Ciscoland When You're Dead," Jan. 2000, http://www.phrack.org/phrack/56/p56-0x0a.org/phrack/56/p56-0x0a.

[6] D. Taylor, "Using a Compromised Router to Capture Network Traffic," unpublished technical report, July 2002, http://www.netsys.com/library/papers/GRE_sniffing.PDF.

[7] A.P. Kosoresow and S.A. Hofmeyr, "Intrusion Detection via System Call Traces," *IEEE Software*, vol. 14, no. 5, pp. 35-42, 1997.

[8] K. Jain and R. Sekar, "User-Level Infrastructure for System Call Interposition: A Platform for Intrusion Detection and Confinement," *Proc. Network and Distributed Systems Security Symp.*, 2000.

[9] T. Garfinkel, "Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools," *Proc. Network and Distributed Systems Security Symp.*, Feb. 2003.

[10] A.T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Fatih: Detecting and Isolating Malicious Routers," *DSN '05: Proc. 2005 Int'l Conf. Dependable Systems and Networks (DSN '05)*, pp. 538-547, 2005.

[11] R. Perlman, "Network Layer Protocols with Byzantine Robustness," PhD dissertation, MIT LCS TR-429, Oct. 1988.

[12] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz, "Listen and Whisper: Security Mechanisms for BGP," *Proc. Symp. Networked Systems Design and Implementation*, Mar. 2004.

[13] S. Kent, C. Lynn, J. Mikkelson, and K. Seo, "Secure Border Gateway Protocol (Secure-BGP)," *IEEE J. Selected Areas in Comm.*, vol. 18, no. 4, pp. 582-592, Apr. 2000.

[14] Y.-C. Hu, A. Perrig, and D.B. Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," *Proc. Eighth ACM Int'l Conf. MobiCom*, Sept. 2002.

[15] B.R. Smith and J. Garcia-Luna-Aceves, "Securing the Border Gateway Routing Protocol," *Proc. Global Internet '96*, 1996.

[16] S. Cheung, "An Efficient Message Authentication Scheme for Link State Routing," *Proc. Ann. Computer Security Applications Conf.*, pp. 90-98, 1997.

[17] M.T. Goodrich, "Efficient and Secure Network Routing Algorithms," provisional patent filing, Jan. 2001.

[18] Y. Cosendai, M. Dacier, and P. Scotton, "Intrusion Detection Mechanism to Detect Reachability Attacks in PNNI Networks," *Recent Advances in Intrusion Detection*, 1999.

[19] Y. Jou, F. Gong, C. Sargor, X. Wu, S. Wu, H. Chang, and F. Wang, "Design and Implementation of a Scalable Intrusion Detection System for the Protection of Network Infrastructure," *Proc. DARPA Information Survivability Conf. & Exposition*, vol. 2, pp. 69-83, 2000.

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

[21] V.N. Padmanabhan and D. Simon, "Secure Traceroute to Detect Faulty or Malicious Routing," *SIGCOMM Comp. Comm. Rev.*, vol. 33, no. 1, pp. 77-82, 2003.

[22] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy, "Highly Secure and Efficient Routing," *Proc. INFOCOM Conf.*, Mar. 2004.

[23] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy, "Amendment to: Highly Secure and Efficient Routing," amendment, Feb. 2004.

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

[25] K.A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R.A. Olsson, "Detecting Disruptive Routers: A Distributed Network Monitoring Approach," *Proc. IEEE Symp. Security and Privacy*, pp. 115-124, May 1998.

[26] J.R. Hughes, T. Aura, and M. Bishop, "Using Conservation of Flow as a Security Mechanism in Network Protocols," *Proc. IEEE Symp. Security and Privacy*, pp. 132-131, 2000.

[27] A. Herzberg and S. Kutten, "Early Detection of Message Forwarding Faults," *SIAM J. Computing,* vol. 30, no. 4, pp. 1169-1196, 2000.
[28] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb, "A Case Study of OSPF Behavior in a Large Enterprise Network," *IMW '02: Proc. Second ACM SIGCOMM Workshop Internet Measurment,* pp. 217-230, 2002.
[29] D. Watson, F. Jahanian, and C. Labovitz, "Experiences with Monitoring OSPF on a Regional Service Provider Network," *ICDCS '03: Proc. 23rd Int'l Conf. Distributed Computing Systems,* p. 204, 2003.
[30] Cisco Systems, "Load Balancing with Cisco Express Forwarding,"http://www.cisco.com/warp/public/cc/pd/ifaa/pa/much/prodlit/loadb_an.pdf, 2006.
[31] Juniper Networks, "JUNOS 6.4 Routing Protocols Configuration Guide,"http://www.juniper.net/techpubs/software/junos/junos64/swconfig64-routing/html/, 2006.
[32] R. Teixeira, K. Marzullo, S. Savage, and G.M. Voelker, "In Search of Path Diversity in ISP Networks," *Proc. ACM/SIGCOMM IMC,* pp. 313-318, 2003.
[33] J. Bellardo and S. Savage, "Measuring Packet Reordering," *Proc. ACM SIGCOMM Internet Measurement Workshop (IMW '02),* 2002.
[34] J.C. R. Bennett, C. Partridge, and N. Shectman, "Packet Reordering Is Not Pathological Network Behavior," *IEEE/ACM Trans. Networking (TON),* vol. 7, no. 6, pp. 789-798, 1999.
[35] G. Almes, S. Kalidindi, and M. Zekauskas, "A One-Way Packet Loss Metric for IPPM," *RFC 2680, IETF,* Sept. 1999.
[36] D. Pullin, A. Corlett, B. Mandeville, and S. Critchley, "Packet Reordering: The Minimal Longest Ascending Subsequence Metric," Feb. 2002, http://www.globecom.net/ietf/draft/draft-critchley-mlas-reordering-00.t%xt.
[37] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser, "Packet Reordering Metric for IPPM," Mar. 2003, http://www.globecom.net/ietf/draft/draft-ietf-ippm-reordering-00.txt.
[38] Cisco Systems, "Detecting and Analyzing Network Threats with NetFlow,"http://www.cisco.com/univercd/cc/td/doc/product/software/ios124/124tcg /tnf_c/ch15/nfhtdt.pdf, 2006.
[39] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM,* vol. 13, no. 7, pp. 422-426, July 1970.
[40] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set Reconciliation with Nearly Optimal Communication Complexity," *Proc. Int'l Symp. Information Theory,* p. 232, June 2001.
[41] "Communications Assistance for Law Enforcement Act of 1994," pub. L. No. 103-414, 108 Stat. 4279, 103rd Congress of the United States of Am.
[42] S. Hanks, T. Li, D. Farinacci, and P. Traina, "Generic Routing Encapsulation GRE," *RFC 1701, IETF,* Oct. 1994.
[43] T.D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM,* vol. 43, no. 2, pp. 225-267, 1996.
[44] A.T. Mizrak, K. Marzullo, and S. Savage, "Detecting Malicious Routers," Univ. of California ay San Diego, technical report CS2004-0789, 2004, http://www.cs.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2004-0789.
[45] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and Secure Message Authentication," *Lecture Notes in Computer Science,* vol. 1666, pp. 216-233, 1999.
[46] GNU Zebra, http://www.zebra.org, 2006.
[47] J.T. Moy, "OSPF version 2," *RFC 2328, IETF,* Apr. 1998.
[48] D.L. Mills, "Network Time Protocol (version 3) Specification, Implementation," *RFC 1305, IETF,* Mar. 1992.
[49] Abilene Network, http://abilene.internet2.edu/, 2006.
[50] User Mode Linux, http://user-mode-linux.sourceforge.net/, 2006.
[51] P. Rogaway, "UMAC Performance (More)," www.cs.ucdavis.edu/~rogaway/umac/2000/perf00bis.html, 2006.
[52] N. Shah, "Understanding Network Processors," Master's thesis, Univ. of California, Berkeley, Sept. 2001.
[53] W. Feghali, B. Burres, G. Wolrich, and D. Carrigan, "Security: Adding Protection to the Network via the Network Processor," *Intel Technology J.,* vol. 06, pp. 40-49, Aug. 2002.
[54] National Institute of Standards and Technology, "Data encryption Standard," *FIPS PUBS 46-3,* Oct. 1999.
[55] National Institute of Standards and Technology, "Advanced Encryption Standard," *FIPS PUBS 197,* Nov. 2001.
[56] N.G. Duffield and M. Grossglauser, "Trajectory Sampling for Direct Traffic Observation," *Proc. ACM SIGCOMM'00,* pp. 271-282, 2000.
[57] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," *Proc. ACM/SIGCOMM,* pp. 133-145, 2002.
[58] K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. Computer Systems,* vol. 3, no. 1, pp. 63-75, 1985.
[59] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems,* vol. 4, no. 3, pp. 382-401, 1982.

Alper Tugay Mızrak received the BS degree in computer engineering from Bilkent University, Ankara, Turkey, in 2000 and the MSc degree in computer science from the University of California, San Diego, in 2002. He is pursuing his Doctor of Philosophy degree in computer science in the University of California, San Diego. His advisors are Stefan Savage and Keith Marzullo. His research interests are computer networks and fault-tolerant distributed systems. He is a student member of the IEEE.

Yu-Chung Cheng received the BS degree from National Taiwan University, Taiwan, in 1998. Since 2001, he has been working toward the PhD degree, advised by Geoff Voelker and Stefan Savage, in computer science at the University of California at San Diego. His research interests are the performance and reliability of wide-area and wireless networks. His thesis focuses on cross-layer 802.11 wireless network benchmarking with multiple vantage points. He coauthored several papers in wide-area networks including one award paper in DSN 2005, and one paper on Web benchmarking by replying TCP flows from Google Web sites.

Keith Marzullo received the PhD degree from Stanford University in 1984. He is a professor in the Computer Science and Engineering Department at the University of California, San Diego where he has been since 1993. Dr. Marzullo works on both theoretical and practical issues of fault-tolerant distributed computing in various domains, including grid computing, mobile computing, and clusters. He is a member of the IEEE.

Stefan Savage received the BS degree in applied history from Carnegie-Mellon University and the PhD degree in computer science and engineering from the University of Washington. He is an assistant professor of computer science and engineering at the University of California, San Diego, and he currently serves as director of the Cooperative Center for Internet Epidemiology and Defenses (CCIED), a joint project between UCSD and the International Computer Science Institute. His research interests lie at the intersection of operating systems, networking, and computer security. More information about him can be found at http://www.cs.ucsd.edu/ savage. He is a member of the IEEE.