Highly Secure and Efficient Routing

Ioannis Avramopoulos, Hisashi Kobayashi, Dept. of Electrical Engineering
Randolph Wang, Dept. of Computer Science
School of Engineering and Applied Science
Princeton University, Princeton, NJ 08544
{iavramop, hisashi}@ee.princeton.edu, rywang@cs.princeton.edu

Arvind Krishnamurthy
Dept. of Computer Science
Yale University
New Haven, CT 06520
arvind@cs.yale.edu

Abstract— **In this paper, we consider the problem of routing in an adversarial environment, where a sophisticated adversary has penetrated arbitrary parts of the routing infrastructure and attempts to disrupt routing. We present protocols that are able to route packets as long as at least one non-faulty path exists between the source and the destination. These protocols have low communication overhead, low processing requirements, low incremental cost, and fast fault detection. We also present extensions to the protocols that penalize adversarial routers by blocking their traffic.**

Key words: security, routing, networking, system design, graph theory.

I. Introduction

Routing failures can disrupt the operation of critical Internet applications. A *fault* in a link or a router (i.e., a node) can be attributed to either benign or malicious causes. Hardware faults, software bugs, and network mis-configurations are examples of the former type, whereas an attacker who penetrates the routing infrastructure is an example of the latter.

It is the responsibility of routing protocols[1] to mitigate the impact of such faults. However, most of existing work on routing has focused on providing robustness when the behavior of faulty components is *fail-stop*. In this paper, we consider faulty components with arbitrary, or *Byzantine*, behavior that is possibly controlled by an adversary.

An adversary or attacker may, for example, inject false routing information into the network, make arbitrary routing decisions, or congest routers by flooding the network with spurious packets. It can also modify, replay, or simply discard packets coming from other routers. Consequently, such a misbehaving router can subvert the routing operation throughout the network [1].

A. Motivation

A routing protocol that is resistant to Byzantine adversaries is important because:

- Despite recent advances in fault-tolerant hardware and software systems, and in software engineering methodologies, the observed behavior of faulty network components can be arbitrarily complex. Coping with such failures at the network layer, in addition to masking such failures at the application layer, may relax the stringent requirements on the underlying hardware and software, and result in more efficient and less costly designs.
- Coping with adversaries is increasingly important as more critical tasks, such as financial, medical, and military applications, utilize the network infrastructure. In such scenarios, it is only safe to treat the behavior of faulty components as Byzantine.
- Strong distributed mechanisms that monitor and maintain connectivity in a highly decentralized global environment may mitigate detrimental effects of strategic conflicts between service providers. For a treatment of the issues that may arise in such a diverse and competitive environment, the reader may refer to [2].

B. Overview

We present protocols that are able to route packets from a source to a destination, provided that a non-faulty path exists between them. The protocols are efficient, in that they (1) can route over a single path, rather than using several paths concurrently,[2] (2) can support links of bandwidth on the order of Gbps at low incremental cost, (3) have low processing requirements on both data and control packets, as they rely on Message Authentication Codes for authentication, and (4) detect faults fast, as faults are detected on a per packet basis, rather than, for example, being detected via a periodic external probing mechanism.

Our main contributions are:

- We synthesize a basic routing protocol with Byzantine robustness using well-known components such as source routing, destination acknowledgements, fault announcements, reserved buffers, and authentication.
- We propose protocol enhancements to reduce the cryptographic computational overheads and also mitigate the adversary's ability to delay packets without being detected.
- We observe that there is a fundamental uncertainty that arises in detecting faults and discuss how this uncertainty reduces the viability of both sharing information regarding faults and blocking traffic from faulty nodes.
- We show that sharing fault knowledge is a hard problem in its general form. We then propose efficient methods for deploying fault sharing in a limited form.
- We show that straightforward attempts to block traffic from faulty nodes could have the unpleasant side-effect

Ioannis Avramopoulos and Hisashi Kobayashi are supported in part by the New Jersey Center for Wireless and Internet Security (NJWINS).

Randy Wang is supported by NSF grants CCR-9984790 and CCR-0313089.

Arvind Krishnamurthy is supported by NSF grants CCR-9985304, ANI-0207399, and CCR-0209122.

[1]We use the term in its broad sense to refer to protocols associated with the routing operation.

[2]Multipath routing, as an optimisation, can be supported in a straightforward manner. However multipath routing is not required for correctness.

of blocking non-faulty nodes as well. We then develop correct protocols for blocking traffic from faulty nodes.

In Section II, we discuss related work on the subject. In Section III, we present a routing protocol with Byzantine robustness and detection. In Section IV, we outline attacks against the protocol of Section III and how this protocol copes with those attacks. The possible attacks also in part motivate possible protocol refinements in Sections V and VII. In Section V, we present two additional fault detection protocols. The first improves the performance of the protocol of Section III when no faulty routers are present in the path, while the second further restricts the adversary's ability to inflict harm. In Section VI, we show that the obvious way to share fault detection state among different sources leads to an NP-complete problem that is also unlikely to have efficient approximate solutions. We also present efficient techniques for limited sharing of fault detection state. In Section VII, we present a method to penalize faulty routers by blocking their traffic. Finally, in Section VIII, we conclude and present directions for future work.

II. Related Work

Perlman [3] classifies network failures into two types: (1) simple and (2) Byzantine. A simple failure is one where some network component (consisting of one or more nodes and/or links) simply becomes inoperative, whereas in a Byzantine failure, a component becomes faulty, and yet continues to operate (incorrectly). We say that a routing protocol is *Byzantine robust* if it is capable of delivering any packet from a source to a destination as long as a non-faulty path exists between them. We also say that a routing protocol has *Byzantine detection* if faulty network components can be identified. Perlman proposes two types of secure routing protocols/network layers.

The first type is based on the use of (1) a flooding-based routing protocol, (2) reserved buffers, and (3) digital signatures. Flooding-based routing ensures that a packet will traverse every link and hence reach its intended destination, as long as a non-faulty path exists. Reserved buffers, together with digital signatures, ensure that packets will not be dropped because of congestion of a node by excessive traffic (which may arise, for instance, in a DoS attack): digital signatures authenticate the source of each packet, and a buffer should be specifically allocated to accommodate a packet from its intended source. Thus, this routing protocol of Perlman is Byzantine robust in the sense defined above.

The second type of secure routing protocol by Perlman is based on the use of (1) a link state routing protocol, (2) reserved buffers, and (3) digital signatures. The reserved buffers and digital signatures serve the same purpose as in the first protocol. Unlike the the flooding-based routing protocol, all routers now have explicit knowledge of the network topology. If we assume that there will be no more than k failures in the network, then forwarding a packet over $k+1$ disjoint routes should guarantee successful delivery of the packet. Note, however, that this routing protocol is not Byzantine robust: the existence of a non-faulty path does not necessarily imply the existence of $k+1$ disjoint routes.

Herzberg and Kutten [4] have proposed the combined use of acknowledgements, timeouts, and fault announcements to detect packet forwarding faults and have recognized its potential to detect Byzantine faults. They present one communication-optimal and one time-optimal protocol, as well as protocols that trade off communication and time optimality. The protocols are presented in an abstract model. This model, however, leaves open many issues crucial to its realization. These include issues such as the precise nature of the authentication mechanism, how this protocol copes with replay attacks, and how this protocol copes with DoS attacks. Our basic protocol of Section III addresses these issues.

Herzberg and Kutten also proposed the flooding of *disconnection notifications* when faults are discovered in the system. Our work indicates that naive handling of such notifications could be manipulated by faulty nodes to discredit non-faulty elements. This conclusion is based on the observation that a faulty source could induce a non-faulty node to drop packets in order to guard against replay attacks. On the other hand, for the case where fault notifications are handled properly, with, for example, recipients taking into consideration that the source itself could be faulty, we show that nodes cannot exploit this additional information using tractable polynomial-time algorithms.

Bradley et al. [5] propose a protocol for detecting and avoiding routers that are dropping or mis-routing (but not modifying) packets. It is based on (1) link state routing and (2) the "conservation of flow" principle. The conservation of flow can be tested if we let the routers count the number of bytes which enter and leave their interfaces and they announce this information periodically. Their approach requires that there is at least one good neighbor to an adversarial router that may drop packets. Protection against traditional DoS attacks, where misbehaving routers congest the network, is also not considered. In contrast, our proposed scheme can detect nodes that corrupt packets and can also cope with DoS attacks.

Awerbuch et al. [6] propose a protocol that detects packet forwarding faults and routes around faults. This protocol utilizes Message Authentication Codes in a way similar to that of our protocol. Here, probes and acknowledgements (ACKs) are used not only by the destination but also by intermediate nodes. Only the source can verify the authenticity of the ACKs. An encryption step prevents malicious routers from tampering with individual ACKs, which could otherwise cause non-faulty nodes or links to be identified as being faulty. Protection against traditional DoS attacks, where misbehaving routers congest the network, is not considered. In Section IV we show a vulnerability of this protocol, and we also present an amendment to address the vulnerability.

Other work in secure routing (such as [7], [8], [9], [10], [11], [12]), is about protecting topology (route) discovery that, although important, is not the focus of this paper. We do note that Byzantine nodes could try to appear in valid routing paths during route discovery in order to misbehave during data transfers. Our proposed techniques are intended to be used in conjunction with the secure route discovery protocols to enhance the robustness of the system.

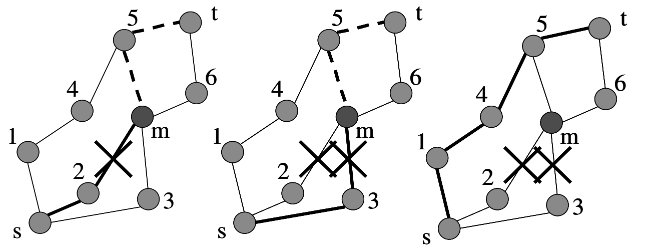

Fig. 1. An example of the operation of the routing protocol of Section III. Source s attempts to communicate with destination t. Router m is malicious and drops packets. The source first attempts to use route $\langle s, 2, m, 5, t\rangle$ to transfer a packet. Both nodes s and 2 set a timeout to receive either an ACK from t or an FA from an intermediate router. Router m drops the packet without informing any other nodes. Router 2 times out when it fails to receive an ACK. It generates and propagates an FA about link $(2, m)$ to s. Node s deletes this link from its topological map and attempts to use another route by calculating the shortest path to the destination in this new map. Eventually, s attempts to use route $\langle s, 1, 4, 5, t\rangle$ and succeeds.

III. Basic Protocol

In this section we present a routing protocol with Byzantine robustness and detection. We first give a definition of what constitutes a faulty component and then justify this definition.

A faulty node is a node that:

- does not follow our protocol, or
- can be impersonated by another node.

The first part of the definition captures a node that is controlled by an adversary or executes buggy code. The second part of the definition is not obvious: we usually associate faults with the notion of malice or harm but, in this case, the behavior of the faulty node is not necessarily malicious or harmful. (The impersonator may be malicious but the impersonated may not be.) A faulty node can be impersonated if, for example, its secret keys used in the routing protocol have been compromised by the adversary through cryptanalysis or other means. Such faulty nodes, even if they operate correctly, are not guaranteed to be able to successfully communicate with other nodes.

A faulty link is a link that:

- drops packets, or
- is incident to a faulty node.

Ideally, we would have like to be able to accurately pinpoint a faulty component. However, from a correctly functioning router, we cannot always tell with certainty whether a link or the downstream router is faulty, although we do not preclude certain cases where this is possible. If a link is detected to be faulty by our protocol, then one or more of the following statements are true:

- The upstream router is faulty.
- The link is faulty.
- The downstream router is faulty.

A. Packet Forwarding with Fault Detection

The packet forwarding protocol utilizes the following mechanisms.

- *Source routing.* The source specifies in every data packet the sequence of nodes that the packet should traverse in order to reach the intended destination. Intermediate nodes read this source-specified route from each packet and forward the packet accordingly.
- *Destination acknowledgements.* The destination of every data packet acknowledges its receipt to the source and every intermediate node. One acknowledgement packet is generated that traverses in reverse direction the path traversed by the corresponding data packet.
- *Timeouts.* The source and every intermediate node set for every data packet a timeout to receive either a destination acknowledgement or a "fault announcement" for this packet.
- *Fault announcements.* When a timeout expires at a node, the node generates a fault announcement (FA) (for the packet triggering the timeout) for the downstream link in the packet's route, and propagates this announcement upstream. The FA is to be interpreted and acted upon only by the source of the data packet. The reason is explained in Section III-C.

Figure 1 illustrates an example of the protocol operation. In this example, a malicious router (m) that drops a packet triggers the generation of an FA by its upstream neighboring router (2). The source (s) responds to the FA by recalculating a route to the destination (t) and eventually succeeds. Pseudocode for the protocol is given in Figure 2.

The simple fault behavior illustrated in the example of Figure 1 exercises only a subset of the mechanisms of the basic protocol. In order to provide Byzantine robustness and detection, we also need the following mechanisms:

- data and control packet authentication,
- a-priori reserved buffers,
- monotonically increasing non-wrapping sequence numbers,
- round-robin scheduling of packet transmission, and
- calculation of appropriate timeout values.

While none of the individual mechanisms of the basic protocol described in this section (III-A) is novel, we note that it is the *combination* of them that delivers the desired robustness and efficiency. We provide the details of some of these mechanisms next.

B. Authentication

Authentication of data packets safeguards against modification and ensures that allocated resources, namely reserved buffers, as explained later, are utilized by the intended sources. Authentication of control packets prevents impersonation: it prevents malicious nodes from forging ACKs and FAs on behalf of non-faulty nodes.

As authentication must be performed for each packet at each node, and the speed of authentication may bound the effective link bandwidth, the performance of the authentication mechanism is crucial. We considered the following alternatives.

- *Digital signatures.* Using digital signatures would have been the most straightforward authentication mechanism, which we have decided against due to its poor performance. Digital signatures may be useful, however, for key setup.

```
// This function is called on reception of an authentic data packet from a
// higher-level protocol by the source.
source( data_packet p)
  p.route = select_route(p.destination);
  p.seqno = seqno++;
  p.auth_tag = authTag(p.route + p.seqno + p.payload, p.route);
  set_timeout(p.route.first_link, p.seqno);
  schedule(p);
//This function is called when the source receives an authentic ACK.
source(ACK a)
  if (timeout_pending(a.seqno))
    cancel_timeout(a.seqno);
  else
    drop(a);
//This function is called when a timeout at the source fires.
source(link e)
  map.delete(e);
//This function is called when the source receives an authentic FA.
source(FA fa)
  if (timeout_pending(fa.seqno) and fa.link is first downstream to fa.source)
    cancel_timeout(fa.seqno);
    map.delete(fa.link);
  else
     drop(fa);
//This function is called when an intermediate node receives an authentic packet.
intermediate(data_packet p)
  source = p.source
  if (p.seqno > source.max_seqno)
    source.max_seqno = p.seqno;
    fa = new FA;
    fa.route = reverse(prefix(p.route, source, this_node)); fa.seqno = p.seqno;
    fa.auth_tag = (fa.route + fa.seqno, fa.route);
    set_timeout(p.seqno);
    schedule(p);
  else
    drop p
//This function is called when an intermediate node receives an authentic ACK.
intermediate(ACK a)
  if (timeout_pending(a.seqno))
    cancel_timeout(a.seqno);
    schedule(a);
  else
     drop(a);
//This function is called when a timeout at an intermediate node fires.
      intermediate(source s, SeqNo n)
        fa = retrieve_FA(s,n)
        schedule(fa)
      //This function is called when an intermediate node receives an authentic FA.
intermediate (FA fa)
  if (timeout_pending(fa.seqno) and fa.link is first downstream to fa.source)
    cancel_timeout(fa.seqno);
    schedule(fa);
  else
     drop(fa);
// This function is called on reception of a data packet p by the destination
// after the authenticity of p has been verified.
destination(data_packet p)
  source = p.source;
  if (p.seqno > source.max_seqno)
    deliver(p);
    source.max_seqno = p.seqno;
    a = new ACK;
    a.route = reverse(p.route); a.seqno = p.seqno;
    a.auth_tag = authTag(a.route + a.seqno, a.route);
    schedule(a);
  else
    drop(p);
```

Fig. 2. Pseudocode for the basic protocol of Section III. In the code, FAs are generated upon reception of data packets and scheduled for transmission later, if necessary. The function "authTag" creates an authentication tag whose details are described in Section III-B. The calculation of the timeout values are described in Section III-C.

- *The multicast authentication construction of Canetti et al. [13].* In this construction, each node is associated with a set of keys obtained from a global pool of keys. The probability that all the keys of a given node are covered by a corrupt coalition is kept small and is configurable. The authentication tag computed by a source node consists of a message authentication code (MAC) for each key held by the source and is verified by each recipient using the keys that the source and recipient share. This authentication mechanism, when used to secure data packet forwarding, is vulnerable to an adversary that tampers with only a subset of the authentication tags such that, for example, the tag verifies at the first downstream router but doesn't verify at subsequent routers, resulting in imprecise fault detection. Furthermore, as Canetti et al. claim, the main cryptographic savings of this mechanism are with respect to authentication tag verification, whereas the limiting factor in secure data packet forwarding is authentication tag generation, since ACKs and FAs must be generated per packet.
- *Tesla [14].* Tesla is a broadcast authentication protocol that relies on loose clock synchronization and delayed key disclosure. In Tesla, messages are first transmitted with authentication tags that contain a MAC computed using an as yet undisclosed key. The key is then subsequently disclosed, at which point the messages that were sent earlier could be verified. Tesla keys are elements of one-way hash chains that are used to authenticate released keys. Using Tesla as the authentication mechanism is an open problem. We identify the following limitations that must be overcome for Tesla to be a viable solution for secure data packet forwarding:
  - Delayed authentication is vulnerable to a DoS attack where a malicious router floods a victim router with spurious MACs, exhausts the victim's memory resources, and causes legitimate packets to be dropped.
  - If two nodes have not communicated securely for a substantial period of time, then the nodes do not have recent enough Tesla keys to efficiently authenticate newly released keys. The system would then incur either a heavy computational overhead for authenticating keys or a large communication overhead corresponding to periodic flooding of Tesla keys.
- *MACs based on pairwise secret keys.* This is the authentication mechanism that we adopt. Its details are described next.

Our scheme requires a secret key for every pair of nodes. The authentication tag for a message comprises of a sequence of MACs computed using the keys shared between the source and each one of the nodes in the source-determined path to the destination. Figure 3 illustrates the structure of the authentication tag. Given a path $\langle s, \ldots, n_i, n_{i+1}, \ldots, t \rangle$, the computation of the MAC for node n_i receives as input both the message and the MACs for nodes $n_{i+1}, \ldots, t$. MACs are therefore computed sequentially from destination to the first intermediate node. The same structure is used for data packets, ACKs, and FAs.

If the computed MAC of each node only included as input the message, then a malicious router could trigger an FA for a non-faulty link. For example, in Figure 3, if n_1 were a malicious router and tampered with the MAC of n_3, then n_3 would have dropped the packet as not authentic, and consequently, n_2 would have generated an FA for link (n_2, n_3), although none of s, n_2, n_3 and physical link (n_2, n_3) are faulty. As a result of using our MAC structure, however, if n_1 tampers with the MAC of n_3, the tampering is detected by n_2, which drops the packet. Consequently, link (n_1, n_2) and/or link (s, n_1) are invalidated, as both are faulty.

Packet processing mainly consists of verifying the authentic-

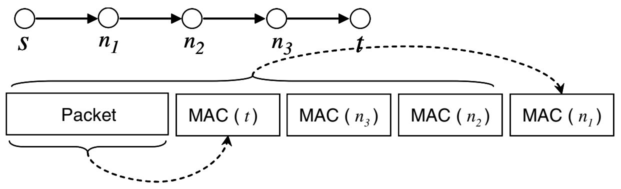

Fig. 3. Computation of authentication tags. Source s sends to destination t via the path $\langle s, n_1, n_2, n_3, t\rangle$. Each receiving node needs to verify the authenticity of the packet. The computation of the MAC for n_i receives as input the secret key that s shares with n_i, the packet (or message), and the MACs for $n_{i+1}, \ldots, t$.

ity of the packet and generating either an ACK or an FA. The ACK or the FA can be generated immediately upon reception of the packet, possibly by a dedicated processor on the link that the packet arrived from. FAs can then be scheduled for transmission later, after the corresponding timeout has fired.

If we restrict the maximum permissible path length to ten hops, then at most eleven MAC computations are required. The upper bound on the network bandwidth is then calculated by setting the time to receive one packet to the time of calculating eleven MACs. We consider some quantitative examples of the link bandwidth that this authentication scheme can support by assuming that there is dedicated hardware for computing the authentication tags. (The incremental cost of assigning a Pentium processor or even special purpose cryptographic hardware to each direction of every link is small when compared to the cost of a Gbps IP router.) The performance of the 64-bit authentication tag UMAC [15] on a Pentium III is 2.2 cycles per byte on 256B packets and 1.2 cycles per byte on 1500B packets. If all packets are of the latter length, then the aforementioned calculation reveals that a 1GHz Pentium III can support a 600Mbps link. If packets are of variable length between 256B and 1000B, then this processor can support roughly a 100Mbps link.

In order to compare the performance with digital signatures, we measured the time that it takes for a 865MHz Pentium III to compute and verify one RSA signature using code from cryptlib 3.0 (http://www.cryptlib.orion.co.nz) and OpenSSL 0.9.7a (http://www.openssl.org/). Signature computation takes approximately 7.6 msec, whereas signature verification takes approximately 0.5 msec. After normalizing the signature computation time to a 1GHz Pentium III, we see that for 1500B packets the upper bound on link bandwidth becomes less than 2Mbps.

C. Reserved Buffers, Timeouts, and Sequence Numbers

When packets are dropped, the fault detection mechanism triggers fault announcements. One of the reasons that routers drop packets is congestion, i.e., when the queues that store packets are full. On one hand, since malicious nodes can incur congestion by overwhelming the network with their own packets, it is desirable to be able to deliver packets despite the presence of such malicious sources. On the other hand, it is desirable to be able to disassociate fault announcements with congestion, since congestion is not inherently a network fault. (We further discuss our planned approach to innocuous packet drops in the future work section of VIII.)

In order to accomplish the aforementioned goals, we employ an a-priori buffer reservation (which ensures that packets are never dropped because of congestion), round-robin scheduling (to minimize the "interference" between sources),[3] and timeouts equal to the worst case round-trip-time to the destination (which attempts to ensure that FAs are not triggered because of congestion). A timeout at the source node can identify whether a path is faulty. Timeouts at intermediate nodes pinpoint the locations of faults.

The choice of the number of outstanding packets allowed per source node involves a trade-off between throughput and recovery time. As the number of outstanding packets increases, throughput also increases. However, an increase in this parameter results in delayed fault detection and, therefore, increased recovery time, because we must use a larger timeout value to allow a larger number of potentially queued packets to drain. Due to round-robin scheduling, an increase in the number of outstanding packets of one source does not affect the timeouts of other sources.

Reserving buffers for sources can be vulnerable to replay attacks if we do not exercise sufficient care: malicious nodes that have stored other sources' previous packets may replay them at a later time and "crowd out" new packets from those sources. The protocol provisions against this attack by utilizing monotonically increasing non-wrapping sequence numbers. When the source inserts a new packet into the network, it also includes in the packet a new sequence number, greater than all the sequence numbers that this source has used before. Furthermore, intermediate nodes maintain, for each incident downstream link and for each source, a window of sequence numbers that the source can legitimately send. (The endpoints of this window are dictated by the source and constitutes a promise from the source that messages will carry sequence numbers only within the specified range.) This window allows the protocol to accommodate out-of-order arrival of the outstanding legitimate packets from a source at any given time, while detecting and dropping illegitimate packets that are due to either replays or faulty sources. A similar window mechanism is also used at the destination.

A ramification of the sequence number-based mechanism is that fault announcements should only be relevant to the source of the packet that triggered the announcement. The reason is that faulty sources can cause packets to be dropped at non-faulty links by, for example, using wrong sequence numbers or dictating incorrect timeout values.

D. Route Selection

Route selection utilizes:

- a topological map,
- fault announcements,
- the number of buffers available to this source at each link,
- link bandwidth, and
- prefix spans, as explained below.

[3]If only one buffer is reserved per source per link, then FIFO scheduling will suffice.

```
INITIALIZE-SINGLE-SOURCE$(G, s)$
{
for each node $v \in V[G]$
do $h[v] \leftarrow 0$
    $d[v] \leftarrow \infty$
    for $h \leftarrow 1$ to $H$
    do $d[v, h] \leftarrow \infty$
        $\pi[v, h] \leftarrow nil$
$d[s] \leftarrow 0$
$h[s] \leftarrow 0$
}

RELAX$(u, v, w, l)$
{
if $h[u] < l(u, v)$ and $d[v] > d[u] + w(u, v)$
then $h[v] \leftarrow h[u] + 1$
        $d[v] \leftarrow d[u] + w(u, v)$
        $d[v, h[v]] \leftarrow d[v]$
        $\pi[v, h[v]] \leftarrow u$
}

PREFIX-SPAN-BELLMAN-FORD$(G, w, l, s)$
{
INITIALIZE-SINGLE-SOURCE$(G, s)$
for $i \leftarrow 1$ to $H$
        do for each edge $(u, v) \in E[G]$
                do RELAX$(u, v, w, l)$
}
```

Fig. 4. Pseudocode of the modified Bellman-Ford shortest path algorithm.

In particular, the links corresponding to valid fault announcements are deleted from the topological map of the source. (Mechanisms for restoring such links are part of ongoing research (Section VIII).) Links that lack available buffers for this source due to currently outstanding packets (packets that have been neither acknowledged nor timed out) are also temporarily deleted from the topological map until buffers become available again. We run a *shortest path algorithm* for the graph consisting of the remaining links.

In Section III-B, we mentioned that by restricting the maximum path length, we can guarantee a certain link bandwidth, given a certain MAC-based authentication speed. Furthermore, the shorter the maximum path is, the greater the link bandwidth that can be supported for a given processor speed. Note that the path length restriction pertains to a restriction on the prefix of a path (as intermediate nodes are burdened with the task of generating FAs that traverse the prefix of a path). Different links are, therefore, capable of supporting different prefix "spans" and can, therefore, be employed at different distances from the source along paths to a destination. The use of prefix spans is clearly desirable for maximizing the throughput of packets sent through a link, but the scheme trades-off reliability as it prevents certain links from being used by sources that are far away from the link, thereby reducing the number of usable paths in the system.

We present an algorithm based on the Bellman-Ford shortest path algorithm that calculates shortest paths in a network where the links have different bandwidths and prefix spans. Bandwidth is factored in the computation by setting the weight or distance on an edge to be the inverse of the bandwidth of the link. We first define the problem formally.

We are given a directed graph $G(V, E)$, with a distance function $w : E \rightarrow \mathcal{R}^+$ and a prefix span function $l : E \rightarrow \mathcal{Z}^+$. The distance of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the distances of its edges. Let $h_p(v_0, v_i)$ denote the number of hops from node v_0 to node v_i in path p. We say that path p respects the prefix span function l if for every $i = 1, \ldots, k$, $h_p(v_0, v_i) \leq l(v_{i-1}, v_i)$.

We now give the pseudocode in Figure 4 for an algorithm that, given a graph G, functions w, l, and a source node $s \in V$, calculates the shortest paths to all destinations that respect l. In the pseudocode, H is the maximum prefix span over all edges.

The complexity of the algorithm is equal to that of Bellman-Ford, i.e. $O(H \cdot |E|)$ and, since $H < |V|$, also $O(|V| \cdot |E|)$. The correctness proof, which we omit for brevity, is similar to that of Bellman-Ford [16]. The intuition is that since Bellman-Ford calculates shortest distance paths for all hops [17], it can be modified to calculate prefix span shortest paths by forcing relaxations on edges to respect the prefix spans.

IV. The Adversary

In this section, we review various attacks that an adversary may mount to prevent communication and how our protocol copes with these attacks. (In general, the types of attacks against a protocol depend on the details of the protocol.) Our protocol is designed to withstand these attacks so that it can continue to deliver packets as long as a non-faulty path exists. We also discuss the extent to which an adversary can impede, rather than prevent, communication.

The adversary can create spurious unauthenticated traffic. We require authentication to work at line speed, and therefore, unauthenticated traffic cannot block authenticated traffic at non-faulty routers.

The adversary can create spurious authenticated traffic. Such spurious traffic cannot block authenticated traffic from non-faulty sources at non-faulty routers, since non-faulty sources are ensured buffers (through a-priori reservations) and link bandwidth (through round-robin scheduling).

The adversary can replay authenticated traffic that has originated from other non-faulty sources. Such replayed traffic cannot block pending authenticated traffic from non-faulty sources, since pending traffic carries sequence numbers that are larger than those of replayed traffic and priority is given to packets with larger sequence numbers.

The adversary can mis-route packets. Mis-routed packets are dropped at the next non-faulty router, if the router does not appear in the source-specified path. The adversary could, however, mount the following form of attack. The adversary, without obstructing the normal flow of a packet, could create a copy and route it through a faster detour (consisting possibly of adversarial nodes only) to the destination. The destination would then reply with an ACK that is sent in the reverse direction of the source-specified path. The router that is adjacent to the destination will drop the ACK since the router has not seen the data packet for which the ACK is being sent. When the original packet reaches this router, it would forward this packet to the destination, but the destination would drop the packet as a duplicate, thereby causing its neighbor to generate an FA about a non-faulty link. This behavior is clearly undesirable. If nodes can authenticate the transmitter of a packet (in a fixed infrastructure network, for example, by the interface that the packet arrived from, or with a MAC computed with a secret key that the transmitter and receiver share), then the above attack scenario could be defeated.

Note: In the protocol proposed by Awerbuch et al. [6] we believe that the encryption step in the computation of the authentication tag of a packet at the source protects against this attack. This protocol is vulnerable, however, to the following attack. The adversary, without obstructing the normal flow of a packet, sends a spurious ACK to a non-faulty router that has already forwarded the packet. If the non-faulty router has no means to verify the identity of the originator of the ACK (which is the case in an ad hoc network that is lacking a protection mechanism), it will accept the spurious ACK and later drop the legitimate ACK as a duplicate. The final consequence is that the source will detect the (non-faulty) link that is incident on the aforementioned non-faulty router as faulty. This vulnerability can be amended if transmissions (possibly between probed nodes only) are protected with a MAC, with such protection also serving as an alternative solution to the encryption step.

The adversary can modify packets. Modifying the content protected by the authentication tag is equivalent to dropping the corresponding packet. Modifying the MACs of upstream routers has no effect, since those MACs are not further utilized. Modifying the MACs of downstream routers is equivalent to dropping the corresponding packet, as the MAC of the first incident downstream router protects the MACs of all other downstream routers.

The adversary can drop packets. The protocol's correctness in this case, which implies its Byzantine robustness, is argued by the following theorem: a packet transmission from a non-faulty source will result in either the reception of a destination acknowledgement or the deletion of a faulty link at the source's topological map. The proof is easy by induction on the number of links and it is omitted for brevity.

The adversary may generate false FAs in an attempt to cause non-faulty links to be excluded by non-faulty sources. Such false FAs are dropped by non-faulty routers because they cannot pass the MAC-based authentication check.

Being a faulty source, the adversary may use wrong sequence numbers (among other means) in an attempt to cause non-faulty routers to drop packets and to cause FAs to be generated upstream of these non-faulty routers. Such an example is shown in Figure 5. The protocol dictates that FAs are only interpreted and acted upon by sources, so these FAs have no effect at any non-faulty routers. Furthermore, faulty intermediate routers can cause non-faulty routers to generate FAs about non-faulty links by replaying old packets. Such an example is shown in Figure 6. These FAs have no effect at non-faulty routers either since FAs for replayed traffic are dropped at non-faulty routers.

We should also point out that the reception of a destination acknowledgement does not necessarily imply that the destination received the packet. For example, consider the route $\langle s, n, t\rangle$. If n can impersonate node t, then n may reply with an acknowledgement to s that appears to have originated from t. However, the attempt to communicate with faulty destinations cannot result in non-faulty links being deleted at non-faulty sources. Similarly, in the route $\langle s, n_1, n_2, t\rangle$, if n_1 can impersonate node n_2, then n_1 can reply with an FA about link (n_2, t) to s, which does not violate Byzantine robustness since the compromised node n_2 is considered faulty according to our definition.

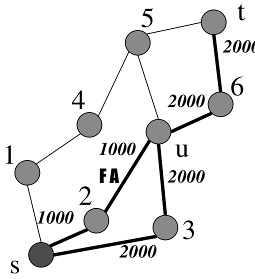

Fig. 5. An example of a faulty source (s) causing a non-faulty router (2) to generate an FA about non-faulty link $(2, u)$ by simultaneously routing two packets with sequence numbers 1000 and 2000 respectively in overlapping routes (we assume that one buffer is reserved per link). Since FAs are only relevant to the corresponding source, Byzantine robustness is not violated.

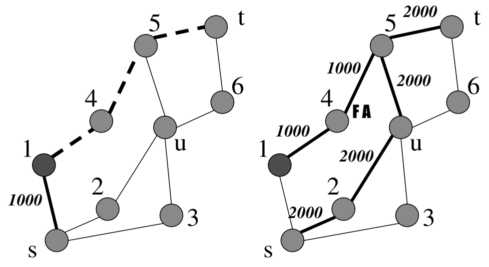

Fig. 6. An example of a faulty (intermediate) router (1) causing a non-faulty router (4) to generate an FA about non-faulty link $(4, 5)$ by first dropping a packet (with sequence number 1000) and then replaying it. Since FAs carry the sequence number of the packet that triggered the FA and FAs for old packets are dropped, Byzantine robustness is not violated.

An adversary may attempt to degrade communication throughput by holding up packets while remaining undetected. In Section III, the timeout parameters are set to the worst case round-trip time when, for example, all routers are fully congested. Such a large value of the timeout, however, allows the adversarial router to emulate local congestion or even congestion in the downstream path without triggering timeouts, thus avoiding detection. In Section V, we present an extension to the protocol of Section III to restrict the adversary's ability to emulate congestion.

The packets that the adversary inserts, possibly optimized to inflict harm, directly affect the goodput (good work) performance of the network. In Section VII, we present a method that attempts to block the adversary. The task is non-trivial as the protocol identifies faulty links rather than routers.

One important measure of performance is the *recovery time* of the protocol, namely, the elapsed time from the moment that communication is interrupted by, for example, packet drops at one or more faulty routers, until the moment that communication is resumed. The recovery time of our protocol is generally small in a moderate size network with sufficient bandwidth, as faults are detected on a per-packet basis, rather than by a periodic external probing mechanism. The recovery time, however, can become significantly worse for certain conditions (such as a large network). Enhancing the protocol to improve recovery time is part of our ongoing research.

V. Protocol Extensions

In this section, we present two protocol extensions. The first one allows us to lower cryptographic overhead and, therefore, also improve communication bandwidth under "normal operations" at the expense of slower fault detection. The second one allows us to limit the ability of a malicious router to delay packets without being declared faulty.

A. Faulty Link Detection by Query

In Section III-B, we have given a MAC-based authentication mechanism that allows us to pinpoint a faulty link. Although much faster than using simple digital signatures, this mechanism requires the computation and verification of one MAC for each node for each data, ACK, or FA packet. These MAC computations bound the communication bandwidth that we can achieve with the basic protocol.

To improve this bound, we introduce an extension to the basic protocol of Section III so that FAs are generated only after a fault has occurred, instead of being generated on a per-packet basis. This modification allows intermediate nodes to compute and verify only one MAC for the data packet and one MAC for its ACK under "normal" fault-free conditions, thus improving communication bandwidth.

Under this modification, we distinguish between two types of packets: "normal" and "query." In normal packets, we retain the source routing, destination ACK and timeout mechanisms. However, when a timeout fires at an intermediate router, this router does not propagate an FA upstream, but only records the packet number and source that resulted in the timeout. When a timeout fires at the source router, the first incident link is not deleted, but a query packet is sent downstream, and a timeout is set for receiving a query ACK or an FA.

Upon reception of a query packet, an intermediate router either drops the query, if the corresponding packet has not been received, or propagates the query downstream and sets a timeout to receive an ACK for the query or an FA. Upon reception of a query packet, the destination either drops the query, if the corresponding packet has not been received, or generates an ACK for the query that propagates upstream. If an intermediate router receives a query ACK, then if the timeout for the corresponding packet has fired, it propagates an FA upstream. Otherwise it propagates the ACK upstream. If the source router receives a query ACK, it deletes the first link in the route from the topological map. If an intermediate router receives an FA, it propagates the received FA upstream. If the source receives an FA, it deletes the link in the FA from the topological map. If the timeout at an intermediate router fires, it propagates an FA upstream. If the timeout at the source fires, it deletes the first link from the topological map. The authentication tag of normal packets and their ACKs, as well as query packets, and their ACKs, and FAs, bears the same structure as that in Section III-B.

In the above protocol, the fault detection time is approximately doubled, as two packets from a source are required to pinpoint one faulty link. As explained earlier, the advantage of this protocol is that in the normal case, where no packet drop occurs, intermediate routers need only to compute/verify one MAC for the packet and one MAC for its ACK.

We proceed to show how to improve performance of normal packets even further. The key observation is that the authentication tag of the packet can be computed on a shorter message than the original message while preserving Byzantine detection. Note that MAC computation of a 64B message is three to four times faster than MAC computation of a 1500B message.

More specifically the source computes a MAC of the source route and sequence number only, for intermediate routers, and a full MAC for the destination. When the packet reaches the destination it may, therefore, have been modified in transit by a malicious router that is present in the path. The destination can, however, detect the modification and, so that the faulty link will be pinpointed, it generates and propagates upstream an ACK that contains a hash of the modified packet. Upon reception of such an ACK, an intermediate router computes a hash of the corresponding packet, which it had previously stored. If the hashes agree, it propagates the ACK upstream. If the hashes don't agree, it drops the ACK, and when the query packet arrives, it generates an FA for its downstream link.

Note that in this extension routers are required to store packets until the corresponding ACK is received or a timeout fires. Due to the use of reserved buffers, this requirement does not impose any additional overhead. If memory consumption becomes an issue, then the hash can be computed on reception of the packet at the expense of extra cryptographic overhead when the path is fault-free. In this case the cryptographic savings are for the source only.

B. Delay Mitigation

A malicious router may intentionally hold onto packets and delay their transmission to emulate the effects of local or downstream congestion (which could be the cumulative effect on multiple downstream routers). To maximize its negative impact, such a router would attempt to introduce a maximum delay but still keep it below the threshold that would trigger a timeout upstream. (If triggered, such a timeout would initiate a route recalculation at the source, which would in turn cause the malicious router to be bypassed.)

While we cannot prevent a malicious router from delaying packets, what we can do is to have enough detailed accounting of delays so that we can pinpoint the locations where excessive delays are being introduced. This is a natural extension of the protocols that we have discussed so far: while the previous discussions focus on pinpointing the locations of packet *drops* and attempts to bypass these locations, this extension focuses on pinpointing the location of excessive packet *delays* and similar attempts of bypass these as well.

Routers measure and record round-trip times (RTTs) of the packets that they forward. Routers also measure and record delays experienced by packets and their ACKs inside these routers. Upon request, routers report recorded RTTs and delays to upstream routers. The request can be a "query" packet similar to that of the above section. Recorded RTTs and delays can be appended in the query ACK. Because these timing statistics reports are protected by the same MAC-based authentication mechanism, a malicious router cannot impersonate another router in its reporting and it cannot tamper with others' reporting. It can, however, falsify reports of delays experienced by packets inside itself. (If none of the routers falsify their reports, it is easy to pinpoint the location(s) of excessive delays and we do not further discuss this case.)

Suppose a router has measured the RTT experienced by a packet downstream as T. It receives a report from downstream routers that indicates that the delay experienced by each downstream router n_i is t_i. After an approximate accounting for wire transmission delays (determined largely by the speed of light), if T exceeds $\sum_i t_i$ "significantly" (by a policy-determined threshold), we conclude that the incident downstream link from this router is faulty, and we propagate an FA upstream from this router. (FAs do not need to bear any information on delays.) This approach of removing routers that falsify delay reports, however, may be unnecessarily harsh (as unnecessary as immediately removing routers that innocuously drop a packet once in a while). For example, a route that experiences falsified congestion and falsified delay reports might still be preferable to a network partition. The same mechanism, namely, a continuous fault metric (in Section VIII), that we plan to introduce to cope with innocuous drops (among other issues), is also applicable to the problem of delay mitigation.

The above protocol requires each router to authenticate the reported delay to every upstream router. We now show how delay mitigation can be achieved with negligible additional cryptographic overhead even if packets are "normal" per Section V-A.

Upon reception of a packet a router, appends to the packet, in an unauthenticated form, the delay that the previous packet experienced inside this router and, furthermore, stores the delays reported by upstream routers. The destination reflects in its ACK the recorded delays. Upon reception of the ACK, a router authenticates the ACK and compares the list of recorded delays in the ACK with the list of stored delays. If they agree, and after a successful comparison with the RTT it forwards the ACK upstream. Otherwise, it drops the ACK and when the query packet arrives it generates an FA for its downstream link.

This technique is generic and permits the establishment of a low-bit-rate stream from downstream routers to the source with small cryptographic overhead that can be used for conveying control information.

VI. Sharing Fault Detection State

Fault announcements are only relevant to the source of the packet that triggered the announcement. The reason is that, as described earlier, faulty sources can trigger fault announcements for non-faulty links. Faulty links are, therefore, independently detected by different sources that attempt to make use of such links. It would be beneficial if fault knowledge could be shared among sources. We first show that in the general case the problem is difficult. We then show two methods for limited sharing of fault knowledge.

A. The Difficulty

One possibility of sharing fault knowledge would be by the following three-stage process:

- Nodes announce links that they have detected as faulty from the fault detection mechanism of Section III.
- Each node maintains a list L of $(node, link)$ pairs, where $(n, l) \in L$ if and only if link l has been announced to be faulty by node n.
- Each node calculates paths that "conform" to their list L. Given two equivalent definitions of the same path: p being a list of nodes $\langle n_1, \ldots, n_k \rangle$, and p' being a list of links $\langle (n_1, n_2), \ldots, (n_{k-1}, n_k) \rangle$, we say p (or p') conforms to list L if $(n, l) \in L$ implies that $n \notin p$ and/or $l \notin p'$.

The correctness of this method is argued by the fact that if node n announces link l as faulty, either n or l is faulty.

- If n is not faulty, then l is faulty, as the fault detection mechanism of Section III only detects faulty links, if employed by a non-faulty source.
- If l is not faulty, then n is faulty: either the fault detection mechanism of Section III had output a fault for a non-faulty link, which is only possible for faulty sources; or n, without necessarily being a source, arbitrarily announced l to be faulty, which is only possible if n is faulty.

The difficulty of employing this method lies in the complexity of calculating conforming paths. In particular, the problem of deciding the existence of a path that connects a given source and destination that also conforms to a given list of fault announcements is NP-complete. The reduction from the Impossible Pairs Path (IPP) problem of [18] is straightforward.[4] We also note that the IPP problem is not only hard to solve, but also hard to find approximate solutions for [19], [20], [21]. Our problem is likely to be as hard to find approximate solutions for as IPP (although we have not verified the conjecture).

B. Efficient Sharing Between Neighboring Routers

Fault knowledge can be efficiently shared between neighboring routers, while preserving the Byzantine robustness property. We propose the following changes to the protocol of Section III.

We call the first link of a route the *lead link*. Detected faults are associated with the lead links traversed by the packets that lead to the corresponding fault announcements: we record a lead link along with the corresponding fault(s), and we say that this lead link *detects* the downstream faulty link(s). Given a lead link (s, r), if it detects a faulty link e, then node s may share with node r this fault knowledge. A router only

[4]In the IPP problem, we are given a source s, a destination t, and a list L of pairs of nodes. The objective is to decide whether a path p consisting of nodes $\langle n_1, \ldots, n_k \rangle$ exists between s and t such that if $(n_1, n_2) \in L$, then $n_1 \notin p$ and/or $n_2 \notin p$.

shares such fault knowledge with the neighbor through which the corresponding faulty link(s) have been detected. Node r, for example, the receiver of the shared fault knowledge, may use this information in its route calculation. Node r deduces that node s, link e, or both must be faulty, and if s is faulty, so is (s, r). Therefore, node r concludes that a fault-free route cannot contain both (s, r) and e. One way of using this information, when a node performs route calculation, is to calculate routes for each of its incident links separately, and for the incident link that is its turn, delete the faulty links that the incident link detects from the topological map, so, for example, during r's route calculation, when it considers the incident link (s, r), it excludes e from consideration.

C. Efficient Sharing Between Non-Neighboring Routers

If router s detects link (u, v) as faulty, it may announce this to u and v. Nodes u and v deduce that node s, link (u, v), or both must be faulty, and, therefore, a fault-free route cannot contain both s and (u, v). Such information can be used in route calculation by calculating routes for each incident link separately, and for the link that is its turn, delete the nodes that have announced this link as faulty. For example, in u's route calculation, when it considers link (u, v), it excludes node s.

VII. Penalizing Faulty Routers

So far, we have focused on avoiding using faulty components—we have not discussed any attempts to block traffic coming from faulty components. There are, however, several reasons why such blockage may be desirable. First, traffic from a faulty node may have an unforeseeable impact on a non-faulty node. Although a design goal of our protocol is to guarantee that a non-faulty node can not be declared faulty by other non-faulty nodes solely because of the actions of faulty nodes, a faulty node might nevertheless try to exploit other vulnerabilities in nodes that are currently non-faulty. For example, traffic from faulty nodes may be able to trigger dormant software bugs in a non-faulty node. Thus, blocking traffic from faulty nodes provides an additional level of protection against such vulnerabilities. Second, traffic from faulty nodes is likely to be a waste of resources. Third, trying to maximize the damage, multiple malicious faulty nodes may well need to communicate among themselves to coordinate their actions. Blocking or reducing their ability to communicate may limit such concerted efforts.

A. Challenges and A Naive Solution

There are at least two challenges in "correctly" penalizing faulty routers. The first challenge is the difficulty of exactly identifying the faulty routers. The fault detection mechanism of Section III only identifies faulty *links*—it does not necessarily pinpoint the exact faulty *routers*. If a link is detected to be faulty by a non-faulty router, then one or more of the following components are faulty: the upstream router, the link, and/or the downstream router. A faulty upstream router may have generated a false FA; a faulty link may be dropping packets; and a faulty downstream router may be dropping or delaying packets. Despite this difficulty, however, as we shall see, it

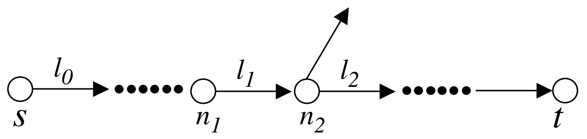

Fig. 7. An example route. Nodes s and t are the source and destination respectively.

is possible to modify the protocol to gradually isolate faulty routers by accumulating fault knowledge over time.

The second challenge in penalizing faulty routers lies in our goal of maintaining Byzantine robustness: if we do not exercise care, the process of blocking faulty nodes may result in the unintended side effect of blocking non-faulty nodes.

We now consider a simple design and show how it fails to achieve this goal. In this design, if a router detects a link to be faulty, it blocks all packets that have the given link at the first position in their source routes. Suppose that s sends a packet with source route $\langle s, \ldots, u, v, \ldots \rangle$. Router v is faulty and drops the packet. Router u generates an FA about link (u, v) that propagates to s. Router s then detects link (u, v) to be faulty and blocks all packets with (u, v) at the first position in their source routes. Suppose now that u sends a packet with source route $\langle u, v, \ldots, r, s, \ldots \rangle$. Router s blocks this packet, and r generates an FA about link (r, s) that propagates to u. Router v lets the FA pass, and therefore u detects link (r, s) to be faulty, although none of s, r, u and (r, s) are faulty. The non-faulty router u will now block packets that originate from non-faulty routers r and s, violating Byzantine robustness.

B. A First Protocol for Blocking Faulty Traffic

We introduce the following changes to the basic protocol of Section III. As in Section VI, we refer to the first link of a route as the *lead link*. If a router detects a link to be faulty, it blocks all packets that have the given link as a lead link and for each such packet it generates and propagates upstream a blocking announcement (BA). Upon reception of a BA, the source deduces that the lead link cannot be used together in a route with the generator of the BA. Route computation should take this knowledge into account by calculating routes for each incident link separately, and for each link that is its turn, delete the generators of BAs for this link. Furthermore, the receiver of a BA should block the generator of the BA, if the generator attempts to use the corresponding link.

C. A Second Protocol for Blocking Faulty Traffic

We again refer to the first link of a route as the *lead link*; we associate detected faults with the lead links traversed by the packets that lead to the corresponding faults; and we say that this lead link *detects* the downstream faulty link(s). Under this new protocol, we introduce two types of packet drops: *inbound drops* and *outbound drops*; furthermore, we also need to modify the route re-calculation algorithm to accommodate these extra packet drops.

Inbound drops: a router blocks (i.e., drops) a packet arriving from an incident link if this incident link detects the lead link traversed by this packet as being faulty. Consider the example topology in Figure 7. If a packet arrives at n_2 via l_1 and l_1 detects l_0 as faulty, n_2 drops the packet. (Intuitively, this blockage makes sense because we know that at least one of l_0 and l_1 must be faulty.)

Outbound drops: a router blocks an arriving packet that intends to leave this router via an incident link if this incident link detects the lead link as being faulty. Furthermore, this router generates an FA and propagates it upstream, declaring this incident link to be faulty. In the example route of Figure 7, if a packet intends to leave n_2 via l_2 and l_2 detects l_0 as faulty, n_2 drops the packet and propagates an FA about l_2 upstream toward s.

Route re-calculation: the protocol must be further modified to account for these extra packet drops. Consider the outbound drop example in the last paragraph. Under the basic protocol of Section III, the FA from n_2 unambiguously signals to s that l_2 is faulty and l_2 should be excluded in any route computation performed by s. Under the new protocol designed to block faulty routers, however, the FA from n_2 only signals that at least one of l_0 and l_2 must be faulty. Responding to this FA, node s can no longer simply exclude l_2 from route calculations. Instead, node s needs to ensure that none of the routes it chooses can contain both l_0 and l_2. One way of ensuring this property is for the source node (such as s) to calculate routes for each of the source's incident links separately, and for the incident link that is its turn (such as l_0), delete from the topological map the link(s) that this incident link detects as faulty (such as l_2).

One may notice that the route re-calculation algorithm above is similar to that given for sharing fault knowledge (Section VI-B). This is not surprising, since in both cases, the source acquires supposed link fault knowledge that is not necessarily true. Indeed, we may integrate the fault knowledge sharing mechanism given in Section VI-B with the route re-calculation mechanism given in this section. This resulting integrated mechanism is different from the mechanism of Section VI-B in only one crucial way: in Section VI-B, the FAs propagated to a source from downstream are unambiguous and only the fault knowledge acquired from neighboring nodes is deemed ambiguous; whereas in the integrated mechanism, *all* fault knowledge, including FAs propagated to a source from downstream and knowledge acquired from neighbors, is deemed ambiguous.

1) The Necessity of FA Generation for Outbound Blockage: In the protocol description, we note that a crucial difference between the way we perform “inbound drops” and “outbound drops” is the generation of an FA in the latter case. To understand why, we consider what would happen if we remove this FA from the protocol in handling the following example.

Suppose that s sends a packet with source route $\langle s, r, \ldots, v, u\rangle$. Router v, which is faulty, generates an FA about link (u, v) that propagates to s. Link (s, r) of router s detects link (u, v) as faulty. Suppose now that u sends a packet with source route $\langle u, v, \ldots, r, s, \ldots\rangle$. Router s will block the packet and r will propagate an FA about link (s, r) to u. Link (u, v) of router u detects link (s, r) as faulty. If r sends a packet with source route $\langle r, s, \ldots, w, u, v, \ldots\rangle$, u will block it, and w will generate an FA about link (w, u) that will propagate to r. Link (r, s) of router r then detects link (w, u) as faulty, although none of routers s, r, w, u and links (s, r), (w, u) are faulty.

D. Protocol Characteristics

In the proposed protocols, if a faulty router r drops all packets of a non-faulty router s, s will block r. The traffic of r will not be affected by other non-faulty routers that have not detected r's misbehavior: this can happen, for example, if r obeys the protocol when handling these other routers' packets. In short, the degree of penalty imposed on a rogue router by the protocol (and the network) is, in some sense, proportional to the degree of its malice.

Whether this gradual and partial blockage is a desirable characteristic depends on one's point of view. In a positive view, one may deem a router that continues to operate correctly for *some* routes, for example, more preferable than a router that is shut down by the protocol immediately upon the first sign of error and therefore works for *no* route. In a negative view, a malicious router, for example, may exploit this characteristic by picking and choosing which entities it wishes to be able to communicate with in order to maximize its negative impact.

The advantage of the protocol of Section VII-C with respect to that of Section VII-B is that it requires less work on the part of the blocker. Inbound drops are performed fast without any cryptographic computations (compare this with the BA generation of the first protocol). In this way FA generation is outsourced to upstream routers that can use the first link of the route for forwarding their own packets. Notice that the work performed on behalf of a router is, in some sense, inversely proportional to the degree of its malice. Outbound drops require the generation of an FA from the blocker. This adheres to the aforementioned principle as well, as the inbound link can still forward packets through the lead link.

The protocol proposed in Section VII-C may degrade recovery time, the elapsed time from the moment that communication is interrupted to the moment that communication is resumed. The reason is that each router may detect the same link as faulty several times, once for each of its incident links. In the protocol of Section VII-B and in the basic protocol of Section III, each router detects a link to be faulty only once. The impact, however, depends on many factors, such as the topology of the network, the location(s) of the fault(s), the number of faulty routers, etc. Improving recovery time is part of our continuing research.

The protocol of Section VII-C may also have a negative impact from a network management perspective. Without the proposed changes, if an FA about link $l = (u, v)$ is delivered to router s, then the network components that need to be investigated are the routers s, u, v, and the link l. With the proposed changes, if a link $l_1 = (u, v)$ has been detected as faulty by link $l_2 = (s, r)$, then the network components that need to be investigated are the routers s, r, u, v and links l_1, l_2.

We should, finally, mention that, in the proposed protocols, as a result of blocking faulty traffic, non-faulty traffic may be temporarily blocked as well. However, Byzantine robustness is not violated and, therefore, if a packet is dropped by non-faulty routers, then it contains at least one faulty link, and, furthermore, route computation will converge to non-faulty routes.

VIII. Summary and Future Work

We have presented a routing protocol with Byzantine robustness and detection. The protocol can be seen as a combination of several components. While none of these is novel by itself, it is the integration of them that is crucial for the correctness and efficiency of the protocol. These components are source routing, destination acknowledgements, timeouts, fault announcements, authentication, reserved buffers, sequence numbers, and round-robin scheduling. Removing any of the components gives the adversary the ability to discredit non-faulty elements and, therefore, cause the Byzantine robustness property to be violated. One reason to stress this observation are the vulnerabilities exhibited by previous systems (some of which we point out) that omit any of the components.

We have discussed route selection and have presented an algorithm that calculates "prefix-span" constrained shortest paths. Links with different prefix span constraints are useful for improving performance at the expense of potentially decreasing reliability.

We have introduced two extensions to the basic fault detection protocol. The first improves performance of "normal" operation at the expense of longer fault detection time. The second restricts the adversary's ability to emulate congestion.

We have shown that sharing fault knowledge is, in general, a difficult problem. We have presented efficient methods of limited sharing of fault knowledge.

We have shown how to isolate/penalize misbehaving routers by blocking their traffic, potentially improving both performance and robustness of the network. This approach, however, may potentially increase fault recovery time. We are continuing research on improving our protocol to reduce recovery time. One promising direction is to leverage the time optimal fault detection protocol of Herzberg and Kutten [4].

One important thread of our ongoing research stems from the need of a mechanism that re-incorporates into the network components that have been previously declared faulty and therefore removed. This issue is also related to how we address innocuous packet drops (drops that are not due to faults). Our proposed solution is to replace a binary faulty vs. non-faulty verdict with a more continuous fault metric: a single packet drop should not result in the outright removal of a link, for example; instead, repeated offenses would gradually degrade the "worthiness" of a link, making it less and less desirable and ultimately removed. A previously removed link, on the other hand, can rehabilitate itself by proving its "worthiness" over time. This continuous fault metric needs to be integrated into the route selection/calculation mechanism and, potentially, even congestion control mechanisms as well.

We are investigating the scalability of this protocol. One line of development is to extend it for the case that the nodes of the network are "Autonomous Systems," rather than routers. We believe that it is feasible to consider secure routes as a class of *policy routes* in the Interdomain Policy Routing Architecture [22], [23]. Such secure protocols would also be candidates for effective protection against Distributed Denial of Service attacks.

Acknowledgments

We would like to thank Prof. Robert Tarjan and Spyridon Triantafyllis for numerous helpful discussions. We would also like to thank Antonis Ellinas for pointing out reference [2]. We would finally like to thank the anonymous reviewers for their helpful comments.

References

[1] P. Papadimitratos and Z. Haas, "Securing the internet routing infrastructure," *IEEE Communications Magazine*, pp. 60–68, Oct. 2002.
[2] Kenneth A. Oye, Ed., *Cooperation Under Anarchy*, Princeton University Press, 1986.
[3] R. Perlman, *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, Massachusetts Institute of Technology, Aug. 1988.
[4] A. Herzberg and S. Kutten, "Early detection of message forwarding faults," *SIAM J. Comput.*, vol. 30, no. 4, pp. 1169–1196, 2000.
[5] K. Bradley et al, "Detecting disruptive routers: A distributed network monitoring approach," *IEEE Network Magazine*, Sept./Oct. 1998.
[6] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *Proc. 2002 ACM Workshop on Wireless Security*, Atlanta, GA, Sept. 2002.
[7] R. Hauser, T. Przygienda, and G. Tsudik, "Lowering security overhead in link state routing," *Computer Networks*, vol. 31, no. 8, pp. 885–894, Apr. 1999.
[8] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," in *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Antonio, TX, Jan. 2002.
[9] S. Murphy and M. Badger, "Digital signature protection of the ospf routing protocol," in *Proc. Symposium on Network and Distributed System Security, NDSS '96*, San Diego, CA, 1996.
[10] Y. Hu, A. Perrig, and D. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," in *Proc. 8th Annual International Conference on Mobile Computing and Networking*, Atlanta, GA, Sept. 2002.
[11] B. Smith, S. Murthy, and J. Garcia-Luna-Aceves, "Securing distance-vector routing protocols," in *Proc. Symposium on Network and Distributed System Security, NDSS '97*, San Diego, CA, 1997.
[12] B. Smith and J. Garcia-Luna-Aceves, "Securing the border gateway routing protocol," in *Proc. Global Internet '96*, London, UK, Nov. 1996.
[13] R. Canetti et al., "Multicast security: A taxonomy and some efficient constructions," in *Proc. IEEE Infocom 1999*, New York, NY, Mar. 1999.
[14] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient and secure source authentication for multicast," in *Proc. Network and Distributed System Security Symposium, NDSS '01*, San Diego, CA, 2001.
[15] *http://www.cs.ucdavis.edu/~ rogaway/umac/*.
[16] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.
[17] R. Guerin and A. Orda, "Computing shortest paths for any number of hops," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, Oct. 2002.
[18] H. Gabow, S. Maheshwari, and L. Osterweil, "On two problems in the generation of program test paths," *IEEE Trans. Software Engineering*, vol. 2, no. 3, Sept. 1976.
[19] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems*, http://www.nada.kth.se/~ viggo/problemlist/compendium.html.
[20] V. Kann, "Polynomially bounded minimization problems that are hard to approximate," *Nordic J. Comp.*, vol. 1, pp. 317–331, 1994.
[21] P. Berman and G. Schitger, "On the complexity of approximating the independent set problem," *Inform. and Comput. 96*, pp. 77–94, 1992.
[22] D. Estrin, M. Steenstrup, and G. Tsudik, "A protocol for route establishment and packet forwarding across multidomain internets," *IEEE/ACM Trans. Networking*, vol. 1, no. 1, pp. 56–70, Feb. 1993.
[23] D. Estrin, Y. Rekhter, and S. Hotz, "Scalable inter-domain routing architecture," in *Proc. ACM SIGCOMM*, Baltimore, MD, Aug. 1992.