A Formal Treatment of Efficient Byzantine Routing Against Fully Byzantine Adversary

Siddhant Goenka
UMBC
sgoenka1@umbc.edu

Sisi Duan
UMBC
sduan@umbc.edu

Haibin Zhang
UMBC
hbzhang@umbc.edu

***Abstract*—We describe efficient path-based Byzantine routing protocols that are secure against *fully* Byzantine adversaries. Our work is in sharp contrast to prior works which handle a weaker subset of Byzantine attacks. We provide a formal proof of correctness of our protocols which, to our knowledge, is the first of its kind. We implement and evaluate our protocols using DeterLab, demonstrating that our protocols are as efficient as those secure against weaker adversaries and our protocols can efficiently and correctly detect routers that fail arbitrarily.**

I. Introduction

In this paper, we revisit the problem of path-based Byzantine message routing, where a strong Byzantine adversary could coordinate corrupted routers to disrupt message delivery.

We study the classic setting for asynchronous networks with bounded delays [8, 18, 21], a setting that captures most of practical communication networks (e.g., Internet, data center networks, ad hoc networks). For these communication networks, the most time and communication efficient end-to-end message routing approach is single-path-based routing, sending messages over a fixed and short path from source to destination. Moreover, there are many additional benefits of path-based routing, such as directly enabling "first-in, first-out" message service [10, 28, 34, 45].

If along the path a link or a router fails, a new (and less optimal) path may be selected. However, it is unnecessarily the case that a new path is always available. Besides, it is equally vital to detect, locate, and recover failures for ensuring long-term network resilience and maintaining delivery efficiency. Therefore, Byzantine routing typically needs to address failure detection.

While Byzantine routing could be also achieved using flooding or multicast [2, 3, 9, 19, 24, 41], these protocols are significantly more expensive in terms of communication complexity. In this paper, we focus our attention on efficient path-based Byzantine routing.

Issues on path-based Byzantine routing. Despite an impressive amount of papers on path-based Byzantine message delivery and Byzantine routing ([5, 7, 15, 28], to cite just a few), these protocols handle only a limited, weaker subset of Byzantine attacks and do not tolerate fully Byzantine adversary that coordinates *multiple* corrupted routers along the path.

978-1-5386-7659-2/18/$31.00 ©2018 IEEE

As the three most representative protocols, ODSBR [7], AKWK [5], and HK [28] assume that the links are "well-behaved" in the sense that the links either work or failures can be detected [11, 25, 49]. This model, however, captures only a rather limited subset of Byzantine attacks. ODSBR, for instance, cannot handle more than one intermittently Byzantine routers. AKWK and HK rely on intermediate routers to detect failures, but do not deal with the scenario that malicious routers can *frame* correct routers.

As another example, Bradley *et al.* [15] describe a router monitoring approach where malicious routers can only fail by dropping or misrouting packets but not modifying packets. Additionally, their approach assumes there is at least one nonfaulty neighbor router to a faulty router that drops packets, which appears to be a very strong assumption that may not hold in general network environments.

Besides, all these protocols are analyzed for independent router failures. None of them formally consider a strong Byzantine adversary that can coordinate multiple corrupted routers to disrupt message delivery without being detected.

To our knowledge, while path-based Byzantine routing has been extensively studied, there are no satisfactory Byzantine routing protocols that are provably secure against arbitrary attacks.

Our approach. Our work distinguishes prior works on path-based routing in two main aspects.

First, in contrast to prior approaches that consider a weaker subset of Byzantine attacks, we levy *no* restrictions on the behavior of an adversary. In our model, the adversary may fail to send the routing messages in time (e.g., dropping packets) or frame other nonfaulty routers. The adversary may coordinate multiple corrupted routers to disrupt routing while keeping faulty routers from being detected. The adversary may behave consistently faulty, or intermittently faulty (to avoid being detected).

Second, while previous works use informal arguments or/and experimental evaluation for the correctness of the protocols, we advocate a rigorous provable-security treatment of Byzantine routing, as other primitives in dependable distributed systems such as Byzantine agreement [33] and Byzantine fault-tolerant state machine replication [17]. To this end, we formalize the security notions under fully Byzantine adversary in the customary sense and provide efficient con-

structions meeting the security notions we defined.

Our contributions. We summarize our contributions as follows:

- We review and analyze the most representative path-based Byzantine routing protocols (ODSBR [7], AKWK [5], HK [28]), and show why they handle only a subset of Byzantine attacks.
- We provide formal security definitions of Byzantine routing that covers general Byzantine attacks. We propose new and efficient Byzantine routing protocols that are secure under the definitions of security we formalized.
- We consider timer setup issues related to our protocols, guiding practical parameter selection.
- We implement and evaluate our protocols. We show that our protocols are at least as efficient as existing ones that are insecure or defend against weaker adversaries. We also show our protocol can efficiently detect various Byzantine failures.

Our protocols in a nutshell. We present three Byzantine routing protocols — BR1, BR2, and BR3. BR1 assumes the source and the destination are nonfaulty, a conventional scenario as in previous works. BR2 handle the case where the destination is Byzantine faulty and may cause correct routers to be falsely detected. To our knowledge, this scenario has not been studied in the literature. BR3 is an efficient hybrid Byzantine routing protocol that improves BR2 in its failure-free scenarios. Both BR1 and BR3 use only symmetric cryptography in their gracious executions.

II. Related Work

Byzantine routing: path-based *vs.* multicast based. Byzantine routing can be generally divided into two categories: single-path-based (or simply path-based) [5, 7, 15, 28] and multicast-based [2, 3, 9, 19, 24, 41].

Path-based routing starts with a single and short path. If a failure occurs along the path, a new path is selected. The protocols can route packets as long as there exists one nonfaulty path between the source and the destination. Our work falls into the category of path-based Byzantine routing.

Instead, multicast-based routing is based on the use of multicast communication (may it be one-to-many or many-to-many communication). While these protocols can be used for applications that require strong reliability, these protocols need significantly higher bandwidth, storage, and system I/O.

Our work is a sharp contrast to prior works on single-path-based Byzantine routing [5, 7, 15, 28, 37] which deal with only a weak subset of Byzantine failures. Meanwhile, these prior practical solutions argue the security of their protocols assuming a single Byzantine failure per path (explicitly or implicitly). We do provide a formal proof considering an arbitrary number of failures in a path.

There are some other secure routing protocols that leverage additional infrastructure [39] or trusted computing base [47]. Obenshain *et al.* [39] built intrusion-tolerant networks that use an additional overlay running on top of multiple IP networks to prevent various attacks. SCION [47] relies on a number of *trusted*, top-tier ISPs in the trust domain to achieve reliable end-to-end communications.

Byzantine routing: external *vs.* internal adversary Byzantine routing relies on message authentication mechanism to prevent packets from being altered. A large number of routing protocols addressed how to achieve message authentication and packet integrity [27, 29, 30, 40, 43, 44].

Using authentication mechanism mainly defends against "external" adversary that oversees and controls the network. However, it is just one step towards secure routing, as there are "internal" routers that can still behave maliciously, e.g., dropping packets, framing other routers, failing to send acknowledgments in time. Addressing internal router failures, however, depend on concrete authentication mechanisms.

BGP and S-BGP. One direction that is somewhat related to routing and Byzantine routing is BGP (border gateway protocol) [42] and S-BGP (Secure-BGP) [32]. BGP manages inter-domain routing on the Internet but uses the hearsay information to update routing tables. S-BGP provides route authentication to prevent malicious routers from injecting false information. One can use digital signatures to authenticate each router along the path. There exist more efficient solutions using aggregated signatures (e.g., [48]).

Aliph Chain, BChain, and Byzantine chain replication. Although not explicitly mentioned anywhere, Byzantine routing shares some similarities with chain-based Byzantine fault-tolerant (BFT) protocols [23, 26, 46], where fully connected servers are organized in a metaphorical chain to exploit pipelined executions for higher throughput. In these BFT protocols, clients requests should be routed to follow exactly the path decided by the leader (the source). Among the three protocols, BChain [23] (adopted in Hyperledger Iroha [31]) achieves Byzantine failure detection for a fully Byzantine adversary. However, there are at least three major differences between the failure detection mechanism in Byzantine routing and the one in BChain. First, Byzantine routing typically assumes that the source is correct, while the source in BChain (and BFT) may be malicious. Second, BChain can work in partially synchronous environments [22], where the bound on message delays may be unknown. (BChain does this via view changes, a mechanism that does not exist in Byzantine routing). Third, different from BChain, Byzantine routing does not in principle need to adhere the exact path decided by the source, as long as either messages can arrive at the destination or failures can be detected.

Secure route discovery. Our Byzantine routing protocol is general and does not depend on any particular Byzantine route discovery protocol. That said, one could combine our protocol with any existing Byzantine route discovery protocol [7, 16, 27, 30, 35, 37, 40, 44].

III. Network and Threat Model

The setting. We consider standard dynamic networks [1, 8],

where the network topology may change over time and nodes and edges may come and go (e.g., Internet, local area networks, ad hoc wireless networks). We consider asynchronous networks with bounded delays [8, 18, 21] as considered in most prior, path-based Byzantine routing protocols [5, 7, 15, 28].

We represent the network using an undirected graph $G = (V, E)$, where V is the set of nodes (routers) and E is the set of edges (links).

For two non-negative integers $a \leq b$, let $[a..b]$ denote the set of integers $\{a, a+1, \cdots, b\}$, and let $[T_a..T_b]$ denote the set of elements $\{T_a, T_{a+1}, \cdots, T_b\}$. We also write $[T_b..T_a]$ (for $a \leq b$) to denote the set of elements $\{T_b, T_{b-1}, \cdots, T_a\}$.

Similarly, let $[p_a..p_b]$ denote of the sequence of nodes $(p_a, p_{a+1}, \cdots, p_b)$. A sequence of nodes $P = [p_0..p_n]$ is a path if $\forall i \in [0..n-1]$, p_i and p_{i+1} are neighbors. In our model, given a pair of source and destination (s, d), the source may select a path $P = [p_0..p_n]$ so that $s = p_0$ and $d = p_n$.

For a router p_i ($i \in [0..n-1]$) in $P = [p_0..p_n]$, we define its *successor* $\vec{p}_i$ as its subsequent router p_{i+1}. For a router a router p_i ($i \in [1..n]$), we define its *predecessor* $\overleftarrow{p}_i$ as its preceding router p_{i-1}. For each p_i ($i \in [1..n]$), we define its *predecessor set* $\mathcal{P}(p_i)$ as $[p_0..p_{i-1}]$. For each p_i ($i \in [0..n-1]$), we define its *successor set* $\mathcal{S}(p_i)$ as $[p_{i+1}..p_n]$.

Threat model. We study *message routing* (i.e., *packet routing*) in the Byzantine failure model.[1] Message routing includes *path selection* and *message forwarding*. Path selection selects (one or many) paths from source to destination, while message forwarding forwards messages to destination over the selected paths. This work considers single-path message routing.

A routing protocol *accepts* a message from some higher level layer in the source, and *delivers* a message to the higher layer in the destination. We may simply say that the source accepts a message and the destination delivers a message.

Our goal is to ensure source to destination message delivery with Byzantine failure detection. Detecting failures in Byzantine routing enables correct and more efficient message delivery, and enables failure isolation and recovery for achieving long-term robustness.

We consider a Byzantine adversary that may coordinate multiple Byzantine nodes to disrupt end-to-end communication without being detected. The Byzantine attacks considered in the paper are ones in the *customary* sense, just as in Byzantine fault-tolerant protocols [17, 33]. As mentioned in §II, the Byzantine attacks include external ones targeting message authentication, and internal ones where an adversary deviates the protocol arbitrarily, including dropping packets, framing other routers, etc.

The attacks considered here are *not* exhaustive. For instance, we do not address denial of service (DoS) attacks and BGP hijacking. We comment that there are known techniques that can be used to mitigate the attacks, as discussed in [5, 39].

[1]In the paper, we use message and packet interchangeably.

We assume "minimum" network connection: there exists at least one path from source to destination.[2]

We assume that 1) every router in the network has a certified public/private key pair and 2) every two routers share a pairwise key for MAC (message authentication code). We assume that the attacker cannot break the cryptographic mechanisms used in the paper. Jumping ahead, our protocol does not use expensive asymmetric cryptography in common, failure-free scenarios.

We do not specify any key distribution algorithms. The key setup can be easily realized by standard infrastructures and mechanisms.

We also do not specify any particular Byzantine route discovery protocol. Our protocol is compatible with existing route discovery protocols. Note that Byzantine routers could act normally during route discovery protocols but later exhibit malicious behavior during the routing stage [5].

For ease of presentation, we do not explicitly consider buffering and packet loss rate. (We do consider this for our implementation.) We assume that the routers have adequate storage and computational capacities, and the network has adequate bandwidth. When we say "a node p does not receive an acknowledgment from another node p' in time," we actually mean "a node p keeps track of the number of acknowledgment message losses from p', and the losses exceed the tolerable threshold before the timer expires."

Definitions of security. We define the following definitions of security for a secure Byzantine routing. All the definitions should be satisfied for an arbitrary network with faulty routers and links and for every execution of the routing protocol [28].

- Liveness: If the source and the destination are nonfaulty, the messages accepted at the source will be eventually delivered at the destination.
- Safety: If the source and the destination are nonfaulty and the destination delivers a message, the message was accepted at the source.
- Detection: If the path from source to destination has failures, at least one failure will be detected.
- Accuracy: If failures (either link or router failures) are detected for an execution of the protocol by the source, the detected links or routers were faulty for the execution.

Complexity measures. The time complexity is the worst-case value over all executions of the time since a message is accepted at the source until either a message is delivered at the destination or a failure is detected. The communication complexity is the worst-case value over all executions of the number of messages transmitted by nonfaulty routers. Note that the time complexity is computed when the source and the destination are both nonfaulty, and the communication complexity should not count the messages sent by faulty routers.

[2]We note that while there are works that ensure message delivery even if there may not be a moment when a correct path from the source to the destination exists, they all rely on expensive multicast-based (flooding-based) routing.

IV. Reviewing and Attacking Prior Path-Based Byzantine Routing Protocols

We review three representative path-based Byzantine routing protocols, and demonstrate why they fail to deal with some active Byzantine failures.

We begin with a protocol using a simple time-out mechanism (ODSBR [7]), then a protocol with a more complex time-out approach (AKWK [5]), and then one that carries the time-out idea "to the extreme" (HK [28]).

ODSBR. ODSBR [7] relies on acknowledgements and probes to detect failures. In ODSBR, the source sets up a timer for a path. If it does not receive an acknowledgement from the destination before the timer expires, the source knows there is a failure and it starts to locate the failure. To this end, the source issues a number of probes to *gradually* identify the failure. Specifically, ODSBR embeds "probes" in the packets. The probes specify the list of intermediate nodes that need to send acknowledgements in addition to the destination.

We first describe ODSBR's authentication mechanism — the MAC chain. Given a path $P = [p_0..p_n]$ (where p_0 is the source and p_n is the destination), for $i \in [1..n]$, let k_i be the key shared between p_0 and p_i. To route a message M over the path P, p_1 computes for the message M a *MAC chain* $[\sigma_n..\sigma_1]$, where the MACs are computed sequentially from the destination to the source: $\sigma_n = \text{MAC}_{k_n}(M)$, $\sigma_{n-1} = \text{MAC}_{k_{n-1}}(M, \sigma_n)$, $\sigma_{n-2} = \text{MAC}_{k_{n-2}}(M, \sigma_n, \sigma_{n-1})$, $\cdots$, $\sigma_1 = \text{MAC}_{k_1}(M, [\sigma_n..\sigma_2])$. Upon receiving a MAC chain, an intermediate node p_i verifies and strips the corresponding MAC σ_i, and sends the remaining message $(M, [\sigma_n..\sigma_{i+1}])$ to its successor $\vec{p}_i$. To distinguish instances of execution, M includes the path information and a unique sequence number.

Note that the MAC for each node p_i to verify in the path depends on the MACs of its successor set $\mathcal{S}(p_i)$ in the path. Assuming the length of the MAC is l, the length of the MAC chain to be verified by successive routers over the path is from nl to l.

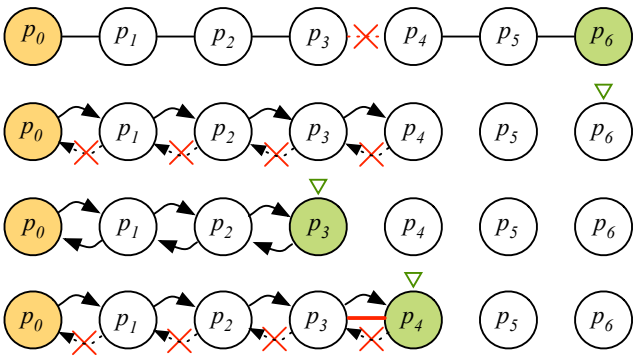

Fig. 1. Failure detection in ODSBR.

Figure 1 demonstrates how ODSBR works using an example for a path $P = [p_0..p_6]$. Suppose the link between p_3 and p_4 is faulty. If the source does not receive an acknowledgement in time, a binary-search like probing mechanism kicks in. The source first adds a probe in the middle of the path (i.e., p_3), expecting an acknowledgement from p_3. In this case, a response will reach the source, as the nodes at the first half of the path (i.e., p_1 and p_2) are all nonfaulty. The source then adds a probe in the middle of the second half the path (i.e., p_4), which will fail to respond. The source now knows the link between p_3 and p_4 is faulty.

ODSBR relies on symmetric MACs only, and is highly efficient in its failure-free scenarios. In case of a failure, ODSBR takes $\Theta(\log n)$ probes to identify the faulty link. Therefore, the probing mechanism becomes rather time-consuming and bandwidth expensive.

Besides the efficiency issue, ODSBR fails to address the following Byzantine attacks: 1) ODSBR only considers Byzantine link failures that are "well-behaved": the links either work well or their failures are detected.[3] 2) ODSBR does not explicitly mention how to detect multiple Byzantine failures within a path.

It is easy to give simple but effective attacks on ODSBR. Let's consider a path $P = [p_0..p_7]$ where p_2 and p_5 are Byzantine faulty. If p_2 and p_5 take turn to exhibit faulty behavior by not forwarding messages or sending acknowledgements, the probes become useless and no failures can be identified.

AKWK. AKWK [5] uses a more involved time-out mechanism, and can therefore detect failures more efficiently compared to ODSBR. In AKWK, *each* node, not just the source, needs to set up a timer. Let Δ be the time-out bound on the transmission delay between two connected neighbor routers. Each node p_i (where $i \in [0..n-1]$) sets the delay as $2(n-i)\Delta$. If a message is sent by p_i over the path P at a time t, expecting an acknowledgement before $t + 2(n-i)\Delta$. Like ODSBR, AKWK uses the same authentication mechanism (i.e., the MAC chain).

While AKWK is more efficient than ODSBR, AKWK has a number of issues, as discussed below.

Issue 1. The main detection idea in AKWK is simple: if a router does not receive the corresponding acknowledgement message in time, the router issues a fault announcement message for its successor. The source will mark the suspected node as faulty. Therefore, AKWK uses the same "well-behaved" link assumption as in ODSBR.

Issue 2. AKWK does not address *framing* attacks. What if in AKWK some faulty router deliberately frames its successor even if the successor behaves correctly? In this case, some nonfaulty routers will be falsely detected.

It is important to note that ODSBR does not have to address this type of attacks, as the source is the only one which can decide which intermediate routers are malicious.

Issue 3. As in ODSBR, AKWK does not explicitly address multiple failures in a single path. It is unclear how the AKWK protocol handles such scenarios.

Issue 4. There are other issues in AKWK. In the originally proposed scheme [5], fault announcement messages are issued

[3] In fact, the assumption dates back to 1980s [11, 25, 49].

by intermediate nodes using the MAC chain. However, the authors of AKWK pointed out in an amendment of the original publication [6] that the nodes issuing fault announcements might be malicious, causing correct nodes to be falsely accused. In the same amendment, the authors proposed a solution using hash functions. There is, however, no security proof of the amended proposal. There is even no description on what security properties the underlying hash function should have. While we cannot verify the correctness of the scheme, we find that the amended solution does not address issues 1−3.

HK. Herzberg and Kutten [28] studied two main protocols, including one that has the same time-out mechanism as AKWK. Here we focus on the second protocol (which we term HK), which uses a more tight time-out check to enable more timely failure detection.

Let's first consider the mechanism as in AKWK, where p_i disconnects from $\vec{p}_i$ if it does not receive the acknowledgment in $2(n-i)\Delta$. To ensure a more timely detection, HK takes a more aggressive approach: once receiving a message package, each router needs to immediately send acknowledgements to *all* predecessor routers over the path. If a router p_i does not receive any of the acknowledgments from $k = i+2, i+3, \cdots, n$ after $2(k-i)\Delta$, it knows it gets disconnected from its successor. The more timely detection comes at the price of much more messages.

Like AKWK, HK relies on the assumption that routers are well-behaved. Moreover, it does not address the issue when there are more than one fully Byzantine routers. We also comment that HK does not specify the authentication mechanism used, and it is unclear how to instantiate HK efficiently.

V. The Protocols

In this section, we present our Byzantine routing protocols — BR1, BR2, and BR3. BR1 and BR2 handle the case where the destination is nonfaulty and Byzantine faulty, respectively. BR3 is an efficient hybrid Byzantine routing protocol that improves BR2 in its failure-free scenarios.

We use MAC chain as those of ODSBR and AKWK [5, 7]), and we now give a detailed description.

Given a path $P = [p_0..p_n]$ (where p_0 is the source and p_n is the destination), for $i \in [1..n]$, let k_i be the key shared between p_0 and p_i. To route over the path P a message M which includes the path information and a sequence number, p_1 computes for the message M a *MAC chain* $[\sigma_n..\sigma_1]$, where the MACs are computed sequentially from the destination to the source: $\sigma_n = \text{MAC}_{k_n}(M)$, $\sigma_{n-1} = \text{MAC}_{k_{n-1}}(M, \sigma_n)$, $\sigma_{n-2} = \text{MAC}_{k_{n-2}}(M, \sigma_n, \sigma_{n-1})$, $\cdots$, $\sigma_1 = \text{MAC}_{k_1}(M, [\sigma_n..\sigma_2])$. Upon receiving a MAC chain, an intermediate node p_i verifies and strips the corresponding MAC σ_i, and sends the remaining message $(M, [\sigma_n..\sigma_{i+1}])$ to its successor $\vec{p}_i$. An example is illustrated in Fig. 2

While ODSBR and AKWK use the same MAC chain mechanism, there is no formal security claim or any proof of correctness. We provide a proof of correctness of the mechanism in Section VI.

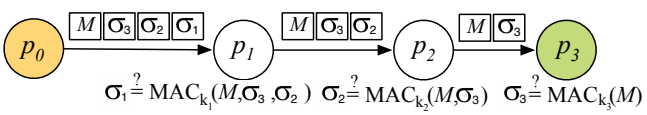

Fig. 2. Message pattern using MAC chain.

A. BR1: Byzantine Routing with Nonfaulty Destinations

BR1 is a path-based Byzantine routing protocol, levying no assumptions on how Byzantine adversary can behave, and capturing adversary that can corrupt multiple routers simultaneously. BR1 is thus a significant improvement to AKWK, correcting all its issues as described in Section IV.

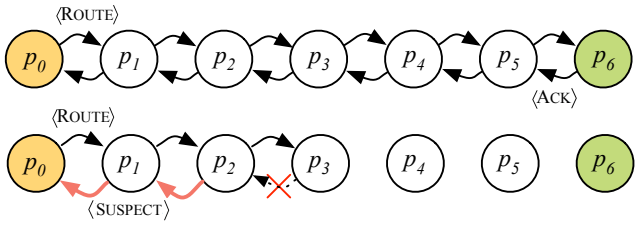

Fig. 3. BR1 message flow. The figure (for the failure case) shows only one possible failure scenario.

Fig 3 depicts the message flow of BR1. BR1 has three types of messages: ⟨ROUTE⟩, ⟨ACK⟩, and ⟨SUSPECT⟩. Initially, a ⟨ROUTE⟩ message is sent by the source along a selected path to the destination. Meanwhile, each node, not just the source, maintains a timer. Upon receiving a ⟨ROUTE⟩ message, an ⟨ACK⟩ message will be sent by the destination. If an intermediate node does not receive the corresponding ⟨ACK⟩ before the timer expires, a ⟨SUSPECT⟩ message will be issued by the node. ⟨ROUTE⟩ and ⟨ACK⟩ use MAC chains, while ⟨SUSPECT⟩ uses digital signatures. BR1 only uses symmetric cryptography, and digital signatures are needed only when a failure occurs. The time and communication complexity of BR1 is proportional to the number of nodes in the path.

We assume that the source knows the topology of the networks. The source maintains a local state st, initially, the network topology. The source takes as input detected failed linked and routers, and outputs a new path.[4] We describe the sequence of steps of BR1.

Step 0: *The source s selects a path $P = [p_0..p_n]$ to the destination d.* The source s takes as input a local state st and outputs a path $P = [p_0..p_n]$ where the source $s = p_0$ and the destination $d = p_n$.

Step 1: *The source computes and sends ⟨ROUTE⟩ with a MAC chain.* Given a path $P = [p_0..p_n]$, p_0 computes for a message M a MAC chain $[\sigma_n..\sigma_1]$. Here, $M = (m, P, seq)$, where m is the message to be sent, P is the ordered nodes, and seq is an increasing sequence number used to distinguish instances of

[4]If equipped with a routing discovery protocol, the state also depends on information collected through routing discovery.

the routing.[5] It then sends its successor p_1 a routing message $(\langle \text{ROUTE} \rangle, M, [\sigma_n..\sigma_1])$. Meanwhile, it sets a timer, expecting an $\langle \text{ACK} \rangle$ message or a $\langle \text{SUSPECT} \rangle$ message.

Step 2: *Intermediate nodes verify and forward* $\langle \text{ROUTE} \rangle$. For a node p_i ($i \in [1..n-1]$), upon receiving $(\langle \text{ROUTE} \rangle, M, [\sigma_n..\sigma_i])$ from p_{i-1}, p_i (with the key k_i) verifies if $\sigma_i = \text{MAC}_{k_i}(M, [\sigma_n..\sigma_{i-1}]$. If it holds, p_i strips σ_i and sends $(\langle \text{ROUTE} \rangle, M, [\sigma_n..\sigma_{i+1}])$ to p_{i+1}. Each time a router p_i sends a message to its successor $\vec{p}_i$, it sets up a timer, expecting an $\langle \text{ACK} \rangle$ message or a $\langle \text{SUSPECT} \rangle$ message.

Step 3: *Destination computes and sends backwards an* $\langle \text{ACK} \rangle$. Upon receiving $(\langle \text{ROUTE} \rangle, M, \sigma_n)$ from p_{n-1}, the destination p_n (with k_n) verifies if $\sigma_n = \text{MAC}_{k_n}(M)$. If it holds, p_n computes an $\langle \text{ACK} \rangle$ message that includes a MAC chain for the *reverse* of the path $P^R = (p_n, \cdots, p_1)$ and sends its predecessor p_{n-1} a message $(\langle \text{ACK} \rangle, seq, [\tau_0..\tau_{n-1}])$.

Step 4: *Intermediate nodes verify: forwarding* $\langle \text{ACK} \rangle$ *or issuing* $\langle \text{SUSPECT} \rangle$.

- If p_i receives a valid $\langle \text{ACK} \rangle$ before the timer expires, it cancels its timer, strips the corresponding MAC, and forwards the rest of $\langle \text{ACK} \rangle$.
- If p_i does not receive an $\langle \text{ACK} \rangle$ message or a $\langle \text{SUSPECT} \rangle$ message before the timer expires, it cancels it timer, and signs a $\langle \text{SUSPECT} \rangle$ message accusing its successor $\vec{p}_i$ of the form $(\langle \text{SUSPECT} \rangle, seq, P, p_i, p_{i+1})$. (Note that a node can suspect its successor and its successor only.)
- If p_i receives a $\langle \text{SUSPECT} \rangle$ but no $\langle \text{ACK} \rangle$ before the timer expires, it verifies that 1) it has seq in its buffer, 2) the accuser is the predecessor of the accused, and 3) the signature is correct. If all conditions are satisfied, it cancels its timer, and forwards $\langle \text{SUSPECT} \rangle$ to its predecessor.[6]

Step 5: *The source accepts an* $\langle \text{ACK} \rangle$*, or a failure is detected and a new path is selected.*

- If the source s (i.e., p_0) receives the corresponding $\langle \text{ACK} \rangle$ message before the timer expires, the source cancels its timer and the message is successfully delivered.
- If the source does not receive a $\langle \text{ACK} \rangle$ or a $\langle \text{SUSPECT} \rangle$ message, the source marks p_2 as faulty and marks all the links connecting p_2 as faulty. The source chooses a new shortest path excluding the node and links marked as faulty.
- If the source receives only $\langle \text{SUSPECT} \rangle$ messages[7] the source deals with $\langle \text{SUSPECT} \rangle$ message that is closest to the source, say, p_i accusing p_{i+1}. s marks the link between p_i and p_{i+1} as faulty, and makes all the other inks connecting p_i and p_{i+1} as potentially faulty. The source selects a new shortest path excluding the link between p_i and p_{i+1}, and if possible, the other links connecting the two nodes marked as potentially faulty. If it is not possible, the source selects randomly a short path with links marked as potentially faulty.

We have the following theorem:

Theorem 1. Assuming the unforgeability of the underlying MAC, BR1 satisfies liveness, safety, detection, and accuracy.

Discussion. In the presence of Byzantine adversary, neither MAC chains nor signatures will make the messages follow exactly the path selected by the source. For instance, an adversary may force $\langle \text{ROUTE} \rangle$ messages to skip some nodes over the path, while still allows these nodes' nonfaulty successors to accept the messages. While this seems to create "black holes," it actually causes no harm.[8] Recall that our goal is not to ensure each router has a consistent state but rather to successfully transmit the messages. This is in sharp contrast to chain-based BFT protocols, and one reason why we can design more efficient solutions than the ones trivially obtained from chain-based BFT protocols.

B. BR2: Byzantine Routing with Fault Destinations

The correctness of BR1 relies on the fact both the source and the destination are nonfaulty, which is a typical setting studied in previous works. A malicious destination may, however, cause nonfaulty to be falsely detected, a situation that we need to avoid. To see this, a faulty destination in BR1 may compute an ill-formed MAC chain so that a nonfaulty router will accuse its nonfaulty successor. (Note that it is also possible to relax the assumption that the sources are nonfaulty as discussed in [39] which aimed to mitigate the damage caused by faulty sources.)

We thus propose BR2 to deal with the case of faulty destination. BR2 is identical to BR1 in the message forwarding phase. However, for the reverse direction, the destination instead directly uses signatures to sign acknowledgments.

C. Setting Time-Out Values

This section discusses how to set time-out values of routers in BR1 and BR2.

Let Δ be the time-out bound on the transmission delay between two connected neighbor routers. In our protocols, each node p_i ($i \in [0..n-1]$), according to the specification of our protocols, sets the delay as $2(n-i)\Delta$. Namely, if a message is sent by p_i over the path P at a time t and but no $\langle \text{ACK} \rangle$ or $\langle \text{SUSPECT} \rangle$ message arrives at time $t + 2(n-i)\Delta$, a $\langle \text{SUSPECT} \rangle$ message should be issued.

The above procedure, however, considers only transmission delay but not computational delay. Specifically, the $\langle \text{SUSPECT} \rangle$ messages are signed using public key cryptography. It is computationally more expensive than conventional $\langle \text{ROUTE} \rangle$ and $\langle \text{ACK} \rangle$ messages. With modern platforms, computing a MAC is many orders of magnitude faster than computing a signature. We need to add extra delay δ for nodes generating

[5] M should include a message type; however, for ease of presentation, we choose not to include the message type in M.

[6] Another designing choice that works is that no matter if the timer expires, intermediate routes already forward $\langle \text{SUSPECT} \rangle$ message if the conditions discussed above are satisfied.

[7] Receiving more than one $\langle \text{SUSPECT} \rangle$ messages is possible if the adversary controls the traffic and mounts a "black-hole" attack.

[8] Byzantine black holes or rushing attacks can be harmful to route discovery protocols, where the attacks can result in establishing adversarial controlled paths.

TABLE I

CHARACTERISTICS OF PATH-BASED BYZANTINE ROUTING PROTOCOLS. THE COLUMNS LABELED "FULLY BYZANTINE," "FORMAL PROOF," "BYZANTINE DESTINATION," AND "EFFICIENT INSTANTIATION" SPECIFY IF THE SCHEME TOLERATES FULLY BYZANTINE ADVERSARY, HAS A FORMAL PROOF, TOLERATES BYZANTINE DESTINATIONS, AND HAS AN EFFICIENT INSTANTIATION, RESPECTIVELY. *HK ALLOWS FASTER DETECTION BY SIGNIFICANTLY INCREASING MESSAGE COMPLEXITY BY USING STRONG ASSUMPTIONS ON HOW ROUTERS MAY FAIL.

	Fully Byzantine	Formal proof	Byzantine destination	Efficient instantiation	Failure detection efficiency
ODSBR [7]				√	$\Theta(\log n)$ rounds
AKWK [5]				√	1 round
HK [28]		√			1 round*
BR1 (this paper)	√	√		√	1 round
BR2 (this paper)	√	√	√	√	1 round
BR3 (this paper)	√	√	√	√	2 rounds

and verifying digital signatures. Meanwhile, as we need to consider the worst-case time-out values, we, in fact, need to add extra delay for any nodes *potentially* generating and verifying digital signatures.

In BR1, each node p_i ($i \in [0..n-1]$) sets the delay as $2(n-i)\Delta+(n-1-i)\delta$. In BR2, each node p_i ($i \in [0..n-1]$) sets the delay as $2(n-i)\Delta+(n-i)\delta$. The minor difference (i.e., δ) is due to the fact that the destination in BR2 is potentially faulty and should use digital signatures, while the destination in BR1 is assumed to be nonfaulty and can use MACs.

D. BR3: Hybrid Byzantine Routing

In case of failures, the sources in BR1 and BR2 have to wait until the timer expires and then take actions. However, the time-out values for BR2 are set to be larger ones for the worst case scenario where the destination needs to use signatures. Meanwhile, the performance of BR2 is also affected by using signatures for $\langle$ACK$\rangle$. We thus present BR3, a hybrid Byzantine routing protocol that relies on symmetric cryptography only and is as efficient as BR1 in its failure-free cases.

BR3 has two modes. In mode 1, only the source sets up a timer, and both the source and the destination use MAC chains to send $\langle$ROUTE$\rangle$ and $\langle$ACK$\rangle$ messages. If the source does not receive the $\langle$ACK$\rangle$ in time, the source will switch to mode 2, where the protocol is the same as BR2 and each router, not just the source, needs to set up an appropriate time. The mode is decided by the source, and the mode type is inserted in the packet before applying the MAC chain.

E. Summary

Table I compares ODSBR, AKWK, HK, and our protocols (BR1, BR2, and BR3).

VI. CORRECTNESS PROOF

Proof of Theorem 1: We begin with a lemma on MAC chain and then proceed to our main proof.

Lemma 2. For the direction from source to destination (resp., from destination to source), any modification of a MAC chain by a faulty router p_i will be detected by the first nonfaulty router in p_i's successor set $\mathcal{S}(p_i)$ (resp., in p_i's predecessor set $\mathcal{P}(p_i)$).

Specifically, for the direction from source to destination, if p_i is a malicious router that modifies the MAC chain (either the message part or any MACs in the chain), the probability of the following event is negligible: some nonfaulty routers in $\mathcal{S}(p_i)$ accept the MAC chain, but some following nonfaulty router rejects the MAC chain. Jumping ahead, the lemma actually shows that the use of MAC chain prevents the scenario that some nonfaulty routers could falsely accuse some other nonfaulty routers.

As an example, consider a path $P = [p_0..p_7]$ with p_2 being malicious. If p_2 modifies the MAC chain, then its successor p_3 will reject the message directly. As another example, for a path $P = [p_0..p_7]$ with p_2 and p_3 being malicious, if p_2 modifies the chain and p_3 strips the corresponding MAC (without verifying its correctness) and passes on the remaining MAC chain, p_4 will reject the message.

Proof of Lemma 2: We only need to prove the case for the direction from source to destination, and the proof for the reverse direction is symmetric.

We prove the claim using standard cryptographic reduction. We show that if an adversary $\mathcal{A}$ that corrupts some router p_i managed to modify the MAC chain that can be accepted by a nonfaulty router in it successor set $\mathcal{S}(p_i)$, we can construct another adversary $\mathcal{B}$ that attacks the unforgeability of the underlying MAC scheme.

More formally, $\mathcal{B}$ runs $\mathcal{A}$ and constructs the environment for $\mathcal{A}$. In the beginning, $\mathcal{B}$ has to "guess" which nonfaulty router is the "victim" one that accepts the modified MAC chain. Specifically, $\mathcal{B}$ simulates an environment for $\mathcal{A}$ with a path $P = [p_0..p_n]$, and randomly guesses a victim correct router, say, p_j, where $j \xleftarrow{\$} [1..n-1]$. Then $\mathcal{B}$ generates the cryptographic keys for routers except for the nonfaulty source, the nonfaulty destination, and the nonfaulty victim router. Clearly, $\mathcal{B}$ does not have the key shared between the source and the victim router. When needed, $\mathcal{B}$ uses the keys generated by itself and runs the underlying MAC to simulate the messages along the path P. It is easy to see that the simulation is perfect from the perspective of $\mathcal{A}$.

When $\mathcal{A}$ outputs a modified MAC chain, $\mathcal{B}$ outputs the corresponding MAC that p_j should verify as a forged MAC. Thus, the advantage of $\mathcal{B}$ is at least equal to that of $\mathcal{A}$ provided that $\mathcal{B}$ guesses the right victim router. The probability of the $\mathcal{B}$ guessing the victim router correctly is $1/(n-1)$. Therefore, if the adversary $\mathcal{A}$'s advantage is $\mathbf{Adv}(\mathcal{A})$, then the adversary $\mathcal{B}$'s advantage of forging a MAC is at least

$1/(n-1) \cdot \mathbf{Adv}(\mathcal{A})$.[9] The lemma now follows. □

We are now ready to prove Theorem 1.

Liveness. As our protocol will eventually traverse each possible path and there is at least one nonfaulty path, the message accepted at the source will be sent along a nonfaulty path and eventually delivered at the destination.

Safety. If the destination delivers a message, according to the specification of the protocol, the corresponding MAC was verified correctly by the destination for the specific sequence number and the path. As the source and the destination are nonfaulty, the adversary does not have access to the key shared between them. Therefore, the probability that a delivered message at the destination was not accepted at the source is at most the probability of adversary forging a MAC, which is negligible. It is easy to turn the intuition to a formal proof, which is a simplification of that of Lemma 2.

Detection. It is easy to see that the detection is a liveness property. We distinguish two cases: 1) all the intermediate nodes are nonfaulty; 2) at least one intermediate node is faulty.

For the first case, according to our protocol, each intermediate node can correctly verify the correctness the MAC and forward the message to the destination which will deliver the message.

For the second case, we just need to show that at least one failure will be detected. We divide the faulty nodes into two types: a) Type-I failure: a router that does not behave according to the protocol so that the router's predecessor fails to receive a valid ⟨ACK⟩ message in time. We call this type of failures as timing failures. b) Type-II failure: a router that sends a ⟨SUSPECT⟩ message maliciously, regardless of the correctness of its successor. We call this type of failures as framing failures.[10] It is clear that the characterization includes all possible failure types in the setting of asynchronous networks with bounded delays.

For both Type-I and Type-II failures, a ⟨SUSPECT⟩ message will be issued by some intermediate router and will arrive at its predecessor before any ⟨ACK⟩ message arrives. This is straightforward for the Type-II failures, and we just need to justify the case for the Type-I failure. In fact, with a timing failure, a modified MAC chain or an empty message will be received the faulty node's predecessor. According to Lemma 2, the predecessor will indeed reject the message.

We further consider two cases: there are no malicious routers over the path from the router which issues the first ⟨SUSPECT⟩ message to the source; there are some other malicious routers over the path. For the first case, it is clear the ⟨SUSPECT⟩ message will be received by the source, as all nodes in its predecessor set are nonfaulty and will forward the signed message. For the second case, if a malicious router does not forward the ⟨SUSPECT⟩ message, a new ⟨SUSPECT⟩ message will be issued by the malicious router's predecessor. Note the according to Lemma 2, the above malicious router cannot correctly compute a valid ⟨ACK⟩ message. (Here, the predecessor receives no valid MAC; one may view the empty message as the modified MAC chain, and Lemma 2 easily applies.) Therefore, for both types of failures, a ⟨SUSPECT⟩ message will be received by the source. According to our protocol, the source will mark some links and routers as faulty once receiving any ⟨SUSPECT⟩ message.

Accuracy. In case of failures, BR1 waits for suspect messages to arrive until the timer expires. BR1 handles the ⟨SUSPECT⟩ message that is the closest to the source, and for each execution, BR1 reports one link failure and two detected nodes (which are neighbors). We will show that the detected link was indeed faulty, and at least one of the two detected nodes was indeed faulty.

For various types of Byzantine failures, here we only need to consider failures that exhibit Byzantine behavior for this specific execution.[11] Fixing a path from source to destination, we distinguish three cases: 1) There is a single faulty router over the path that has a timing failure; 2) There is a single faulty router that is a framing failure; 3). There are more than one failures over the path.

Case 1: In this case, the predecessor of the timing failure will issue a ⟨SUSPECT⟩ message which will be forwarded to the source. This includes the scenario where the faulty node sends a modified MAC chain, and according to Lemma 2, the modified MAC chain will be rejected by its successor and a ⟨SUSPECT⟩ message will be issued. The source will mark the link between the faulty router and its predecessor as faulty, and mark both the accuser and the accused as potentially faulty.

Clearly, the link between the faulty router and its predecessor was indeed faulty. And one of the two routers detected, the accused, was faulty.

Case 2: With a framing failure, a ⟨SUSPECT⟩ message is issued even if its successor is actually nonfaulty. In this case, the detected link was faulty, as one of the endpoint routers was faulty. Meanwhile, one of the two routers detected, the accuser, was faulty.

Case 3: Suppose that there are more than one faulty routers which exhibit Byzantine behavior. This may lead to more than one ⟨SUSPECT⟩ messages. However, *only* the ⟨SUSPECT⟩ message related to the faulty node that is closest to the source (which we term as the "last ⟨SUSPECT⟩ message") will be handled, according to our protocol. This essentially allows us to reduce the case 3) to the case 1) and the case 2). Clearly, at least one of the two routers in the last ⟨SUSPECT⟩ message is faulty. If both of the neighbor routers for the last ⟨SUSPECT⟩ message are faulty, both will be marked as potentially faulty.

The theorem now follows. ∎

[9] More formally, our definitions and proof should include a security parameter. and the probability should be taken over the security parameter.

[10] Strictly speaking, there is one more type where a malicious router may send both ⟨ACK⟩ and ⟨SUSPECT⟩ message altogether. If the ⟨ACK⟩ arrives first, then the router appears to behave correctly. Otherwise, the case is identical to the case of Type-II failures. This is because according to our protocol, once a ⟨SUSPECT⟩ message arrives first, all the ⟨ACK⟩ messages will be ignored.

[11] If Byzantine nodes do not exhibit Byzantine behavior, then the messages will be correctly forwarded and the nodes are technically nonfaulty for the specific execution.

VII. Implementation and Evaluation

A. Implementation

Our experiments are carried out using DeterLab [13]. Our test setting comprises a cluster of 50 machines (2.13GHz Xeon processor, 4GB RAM). We create a pseudo-random graph with at most 20 machines over the longest path.

Each machine runs an instance of our application and acts as a router. Our application is written in Java, and the source code is available at: github.com/sid15g/SecureRouting

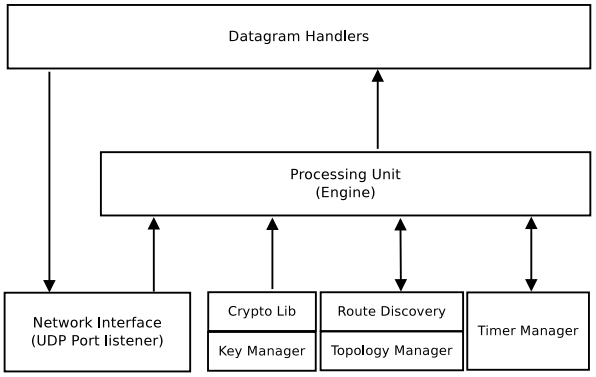

Fig. 4. Application Stack

Fig. 4 depicts our application stack. The Network interface is responsible for sending and receiving datagrams. We use UDP sockets to handle each type of datagram. Packet failures and acknowledgments are handled using our application. While we only implement a UDP-based solution, our system can be extended to use TCP as well.

The topology of our network is maintained using a matrix A, where the entry $A[i][j]$ defines the weight of the link between the node p_i and the node p_j. The source updates the topology state based on the $\langle$SUSPECT$\rangle$ messages received.

We do not implement any particular key distribution algorithm; for simplicity, we hardcode and maintain the keys in a key manager. We use HMAC as the MAC scheme and ECDSA as the digital signature scheme.

Our route discovery implements the shortest path algorithm over the graph maintained by the topology manager. Timer manager maintains timers for messages transmitted. We implement three different handlers for three types of messages: $\langle$ROUTE$\rangle$, $\langle$ACK$\rangle$, and $\langle$SUSPECT$\rangle$. All the actions related to a datagram is defined in its respective handler. Our processing engine maintains a buffer where all the incoming datagrams are pushed. The engine pops the datagrams and processes it sequentially, using the datagram handlers.

B. Evaluation

In our configuration, the average network latency is about 4 ms. We first assessed two settings where the MAC chain is used and cryptography is not used at all, respectively. We observed no obvious performance difference between the two settings. Therefore, the overhead incurred by the MAC chain is negligible. This is because modern hardware can process symmetric cryptography insanely fast. (One thousand 128-bit AES calls can be proceeded around 20 μsec using a commodity Intel processor.) For the same reason, there is no need to compare our work with ODSBR or AKWK in failure-free cases, as their system performance is at best as efficient as that without using cryptography.

We then evaluated the robustness and effectiveness of our protocols in the presence of various failures and attacks. Specifically, we consider the following scenarios: 1) there is one timing failure over the path; 2) there is one framing failure over the path; and 3) there are more than one failures over the path. For all the cases, we injected random failures and tried to see how our protocol reacts. Our protocol can effectively detect the failures within the time-out value set by the source, as analyzed in our protocol and our proof. For *all* the above cases, the source has to wait for the timers to expire to decide the failures, the detection time equals the time-out value of the source. While this shows the robustness of our protocol, there is nothing interesting to report.

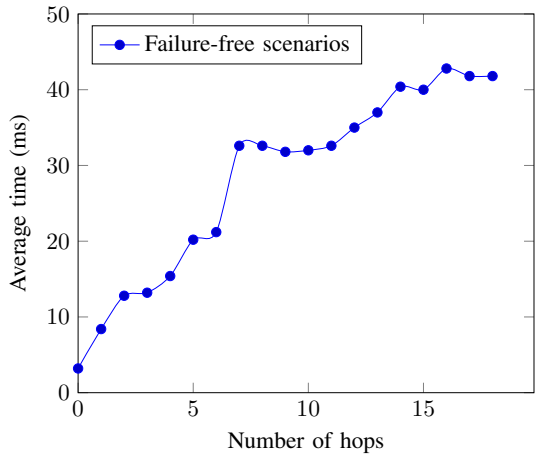

Fig. 5. The latency in failure-free scenarios.

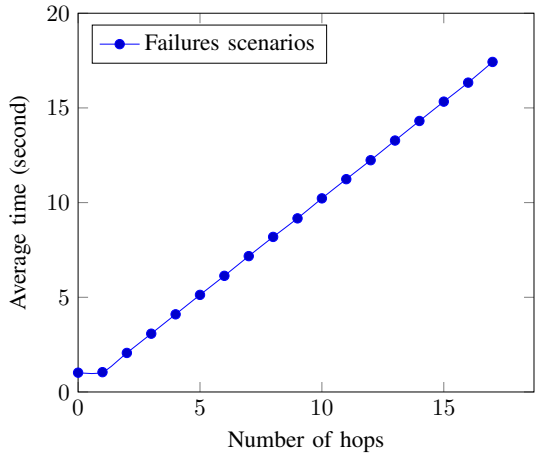

Fig. 6. The latency in failure scenarios.

Last, we reported the average round-trip latency for failure

and failure-free scenarios in Fig. 5 and Fig. 6, respectively. Specifically, for the failure cases, we measured the latency as the time period from the time the source sends a message to the time the source receives the corresponding acknowledgment. This includes the time period that it has to wait to detect a failure for one path and that it selects another path to transmit the same message.

VIII. Conclusion

We revisit the problem of efficient, path-based Byzantine routing. We first analyze representative path-based Byzantine routing protocols and demonstrate why they only deal with a weaker subset of Byzantine attacks. We then devise the first provably secure, path-based Byzantine routing protocols that are secure against fully Byzantine adversary in the customary sense of dependable distributed systems. We implement and evaluate our system, showing our protocols are both highly efficient and robust.

Acknowledgment

Many thanks for the helpful comments from the anonymous NCA referees.

References

[1] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. *FOCS*, 1987.

[2] Y. Afek, B. Awerbuch, E. Gafni, E. Rosen, and N. Shavit. Slide — the key to polynomial end-to-end communication. *J. Algorithms*, 22 (1997), pp. 158–186.

[3] Y. Afek and E. Gafni. End-to-end communication in unreliable networks. *PODC*, 1988.

[4] Y. Amir, P. Bunn, and R. Ostrovsky. Authenticated adversarial routing. *TCC 2009.*

[5] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. *INFOCOM 2004.*

[6] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Amendment to: Highly secure and efficient routing. Available: https://homes.cs.washington.edu/~arvind/papers/amendment.pdf

[7] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens. ODSBR: An on-demand secure Byzantine resilient routing protocol for wireless ad hoc networks. *ACM TISSEC*, vol. 10, issue 4, Jan. 2008.

[8] B. Awerbuch and S. Even. Reliable broadcast protocols in unreliable networks. *Networks*, 16 (1986), pp. 381–396.

[9] B. Awerbuch, O. Goldreich, and A. Herzberg. A quantitative approach to dynamic networks. *POPDC*, 1990.

[10] A. E. Baratz, J. P. Gray, P. E. Green, Jr., J. M. Jaffe, and D. P. Pozefsky. SNA networks of small systems. *IEEE J. Selected Areas in Comm.*, 3 (1985), pp. 416–426.

[11] A. E. Baratz and A. Segall. Reliable link initialization procedures. *IEEE Trans. Comm.*, 36 (1988), pp. 144–152.

[12] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *CRYPTO 1996.*

[13] T. Benzel. The science of cyber security experimentation: the deter project. *ACSAC 2011.*

[14] A. Bessani, J. Sousa, and E. Alchieri. State machine replication for the masses with BFT-SMART. *DSN '14.*

[15] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. IEEE Network Magazine, Sept./Oct. 1998.

[16] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, and N. McKeown. SANE: A protection architecture for enterprise networks. *USENIX Security Symposium*, 2006.

[17] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst*, 20(4): 398–461, 2002.

[18] C. T. Chou, I. Cidon, I. Gopal, and S. Zaks. Synchronizing asynchronous bounded delay networks. *Distributed Algorithms: Second International Workshop*, LNCS 312, Springer, 1988, pp. 212–218.

[19] I. Cidon and R. Rom. Failsafe end-to-end protocols in computer networks with changing topology. *IEEE Trans. Comm.*, 35 (1987), pp. 410–413.

[20] D. Dolev. The Byzantine Generals strike again. Technical Report, Stanford University, Stanford, CA, 1981.

[21] D. Dolev, J. Halpern, B. Simons, and R. Strong. Dynamic fault-tolerant clock synchronization. *J. ACM*, 42 (1995), pp. 143–185.

[22] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM* 35(2): 288–323, 1988.

[23] S. Duan, H. Meling, S. Peisert, and H. Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration.OxtslOPODIS 2014.

[24] S. G. Finn. Resynch procedures and failsafe network protocol. *IEEE Trans. Comm.*, 27 (1979), pp. 840–846.

[25] O. Goldreich, A. Herzberg, and Y. Mansour. Source to destination communication in the presence of faults. *PODC*,1989.

[26] R. Guerraoui, N. Knezevic, V. Quema, and M. Vukolic. The next 700 BFT protocols. *EuroSys 2010*, pp. 363?376, ACM, 2010.

[27] R. Hauser, T. Przygienda, and G. Tsudik. Lowering security overhead in link-state routing. *Computer Networks*, vol. 31, no. 8, pp. 885–894, 1999.

[28] A. Herzberg and S. Kutten. Early detection of message forwarding faults. *SIAM Journal on Computing*, vol. 30, Issue 4, pp. 1169–1196, 2000.

[29] Y. Hu, D. Johnson, and A. Perrig. SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks. *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications (WMCSA).*

[30] Y. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *8th Annual International Conference on Mobile Computing and Networking*, 2002.

[31] Hyperledger architecture, volume 1: Introduction to Hyperledger business blockchain design philosophy and consensus. https://www.hyperledger.org/

[32] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.

[33] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. on Programming Languages and Systems* 4(3): 382–401.

[34] J. M. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *IEEE Trans. Comm.*, 28 (1980), pp. 711–719.

[35] S. Murphy and M. Badger. Digital signature protection of the OSPF routing protocol. *NDSS '96*, 1996.

[36] D. Nicol, S. Smith, and M. Zhao. Evaluation of efficient security for BGP route announcements using parallel simulation. *Simulation Modelling Practice and Theory* 12, 187–216, 2004.

[37] R. R. Obelheiro and J. da S. Fraga. A lightweight intrusion-tolerant overlay network. *ISORC 2006.*

[38] R. R. Obelheiro, J. da S. Fraga. Overlay network topology reconfiguration in Byzantine settings. *PRDC 2007.*

[39] D. Obenshain, T. Tantillo, A. Babay, J. Schultz, A. Newell, M. E. Hoque, Y. Amir, and C. Nita-Rotaru. Practical intrusion-tolerant networks. *ICDCS 2016*, pp. 45–56.

[40] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. *Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2002.

[41] R. Perlman. Network layer protocols with Byzantine robustness. Ph.D. thesis. MIT Laboratory for Computer Science, Cambridge, MA, 1988.

[42] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). RFC1771. http://www.ietf.org/rfc1771.txt, March 1995.

[43] K. Sanzgiri, B. Dahill, B.N. Levine, C. Shields, and E.M. Belding-Royer. A secure routing protocol for ad hoc networks. *ICNP 2002.*

[44] B. Smith, S. Murthy, and J. Garcia-Luna-Aceves. Securing distance-vector routing protocols. *NDSS*, 1997.

[45] A. Tannenbaum. *Computer Networks*, Prentice–Hall, Englewood Cliffs, NJ, 1981.

[46] R. van Renesse, C. Ho, and N. Schiper. Byzantine chain replication. *OPODIS 2012.*

[47] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. Andersen. SCION: Scalability, control, and isolation on next-generation networks. *IEEE Symp. Security and Privacy (SP) 2011.*

[48] M. Zhao, S. Smith, and D. Nicol. Aggregated path authentication for efficient BGP security. *CCS*, 2005.

[49] H. Zimmerman. OSI reference model — the ISO model of architecture for open systems interconnection. *IEEE Trans. Comm.*, 28 (1980), pp. 425–432.