Authenticated Adversarial Routing*

Yair Amir[1], Paul Bunn[2], and Rafail Ostrovsky[3]

[1] Johns Hopkins University Department of Computer Science,
Baltimore, MD 21218, USA. yairamir@cs.jhu.edu
[2] UCLA Department of Mathematics,
Los Angeles, CA 90095, USA. paulbunn@math.ucla.edu
[3] UCLA Department of Computer Science and Department of Mathematics
Los Angeles, CA 90095, USA. rafail@cs.ucla.edu

Abstract: The aim of this paper is to demonstrate the feasibility of authenticated throughput-efficient routing in an unreliable and dynamically changing synchronous network in which the majority of malicious insiders try to destroy and alter messages or disrupt communication in any way. More specifically, in this paper we seek to answer the following question: Given a network in which the majority of nodes are controlled by a node-controlling adversary and whose topology is changing every round, is it possible to develop a protocol with polynomially-bounded memory per processor that guarantees throughput-efficient and correct end-to-end communication? We answer the question affirmatively for extremely general corruption patterns: we only request that the topology of the network and the corruption pattern of the adversary leaves at least one path each round connecting the sender and receiver through honest nodes (though this path may change at every round). Out construction works in the public-key setting and enjoys bounded memory per processor (that is polynomial in the network size and does not depend on the amount of traffic). Our protocol achieves *optimal transfer rate* with negligible decoding error. We stress that our protocol assumes no knowledge of which nodes are corrupted nor which path is reliable at any round, and is also fully distributed with nodes making decisions locally, so that they need not know the topology of the network at any time.

The optimality that we prove for our protocol is very strong. Given any routing protocol, we evaluate its efficiency (rate of message delivery) in the "worst case," that is with respect to the *worst* possible graph and against the *worst* possible (polynomially bounded) adversarial strategy (subject to the above mentioned connectivity constraints). Using this metric, we show that there does not exist *any* protocol that can be asymptotically superior (in terms of throughput) to ours in this setting.

We remark that the aim of our paper is to demonstrate via explicit example the feasibility of throughput-efficient authenticated adversarial routing. However, we stress that out protocol is *not* intended to provide a practical solution, as due to its complexity, no attempt thus far has been made to reduce constants and memory requirements.

Our result is related to recent work of Barak, Goldberg and Xiao in 2008 [9] who studied fault localization in networks assuming a private-key trusted setup setting. Our work, in contrast, assumes a public-key PKI setup and aims at not only fault localization, but also transmission optimality. Among other things, our work answers one of the open questions posed in the Barak et. al. paper regarding fault localization on multiple paths. The use of a public-key setting to achieve strong error-correction results in networks was inspired by the work of Micali, Peikert, Sudan and Wilson [14]

* Full version of the paper is available on-line [5].

who showed that classical error-correction against a polynomially-bounded adversary can be achieved with surprisingly high precision. Our work is also related to an interactive coding theorem of Rajagopalan and Schulman [15] who showed that in noisy-edge static-topology networks a constant overhead in communication can also be achieved (provided none of the processors are malicious), thus establishing an optimal-rate routing theorem for static-topology networks.

Finally, our work is closely related and builds upon to the problem of End-To-End Communication in distributed networks, studied by Afek and Gafni [1], Awebuch, Mansour, and Shavit [8], and Afek, Awerbuch, Gafni, Mansour, Rosen, and Shavit [2], though none of these papers consider or ensure correctness in the setting of a node-controlling adversary that may corrupt the majority of the network.

Keywords: Network Routing; End-to-End Communication; Fault Localization; Error-Correction; Multi-Party Computation; Communication Complexity

1 Introduction

Our goal is to design a routing protocol for an unreliable and dynamically changing synchronous network that is resilient against malicious insiders who may try to destroy and alter messages or disrupt communication in any way. We model the network as a communication graph $G = (V, E)$ where each vertex (node) is a processor and each edge is a communication link. We do not assume the topology of this graph is fixed or known by the processors. Rather, we assume a complete graph on n vertices, where some of the edges are "up" and some are "down", and the status of each edge can change dynamically at any time.

We concentrate on the most basic task, namely how two processors in the network can exchange information. Thus, we assume that there are two designated vertices, called the sender S and the receiver R, who wish to communicate with each other. The sender has an infinite read-once input tape of *packets* and the receiver has an infinite write-once *output tape* which is initially empty. We assume that packets are of some bounded size, and that any edge in the system that is "up" during some round can transmit only one packet (or control variables, also of bounded size) per round.

We will evaluate our protocol using the following three considerations:

1. **Correctness.** A protocol is *correct* if the sequence of packets output by the receiver is a prefix of packets appearing on the sender's input tape, without duplication or omission.
2. **Throughput.** This measures the number of packets on the output tape as a function of the number of rounds that have passed.
3. **Processor Memory.** This measures the memory required of each node by the protocol, independent of the number of packets to be transferred.

All three considerations will be measured in the worst-case scenario as standards that are guaranteed to exist regardless of adversarial interference. One can also evaluate a protocol based on its dependence on global information to make decisions. The protocol that we present in this paper will not assume the internal

nodes have a global view of the network. Such protocols are termed "local control," in that each node can make all routing decisions based only on the local conditions of its adjacent edges and neighbors.

Our protocol is designed to be resilient against a malicious, polynomially-bounded adversary who may attempt to impact the *correctness*, *throughput*, and *memory* of our protocol by disrupting links between the nodes or by taking direct control over the nodes and forcing them to deviate from our protocol in any manner the adversary wishes. In order to relate our work to previous results and to clarify the two main forms of adversarial interference, we describe two separate (yet coordinated with each other) adversaries:[4]

Edge-Scheduling Adversary. This adversary controls the *links* between nodes every round. More precisely, for each round, this adversary decides which edges in the network are up and which are down. We will say that the edge-scheduling adversary is *conforming* if for every round there is at least one path from the sender to the receiver (although the path may change each round).[5] The adversary can make any arbitrary poly-time computation to maximize interference in routing, so long as it remains *conforming*.

Node-Controlling Adversary. This adversary controls the *nodes* of the network that it has corrupted. More precisely, each round this adversary decides which nodes to corrupt. Once corrupted, a node is forever under complete adversarial control and can behave in an arbitrary malicious manner. We say that the node-controlling adversary is *conforming* if every round there is a connection between the sender and receiver consisting of edges that are "up" for the round (as specified by the edge-scheduling adversary) and that passes through *uncorrupted* nodes. We emphasize that this path can change each round, and there is no other restriction on which nodes the node-controlling adversary may corrupt (allowing even a vast majority of corrupt nodes).

There is another reason to view these adversaries as distinct: we deal with the challenges they pose to correctness, throughput, and memory in different ways. Namely, aside from the conforming condition, the edge-scheduling adversary cannot be controlled or eliminated. Edges themselves are not inherently "good" or "bad," so identifying an edge that has failed does not allow us to forever refuse the protocol to utilize this edge, as it may come back up at any time (and indeed it could form a crucial link on the path connecting the sender and receiver that the conforming assumption guarantees). In sum, we cannot hope

[4] The separation into two separate adversaries is artificial: our protocol is secure whether edge-scheduling and corruption of nodes are performed by two separate adversaries that have different capabilities yet can coordinate their actions with each other, or this can be viewed as a single coordinated adversary.

[5] A more general definition of an edge-scheduling adversary would be to allow completely arbitrary edge failures, with the exception that in the limit there is no permanent cut between the sender and receiver. However, this definition (while more general) greatly complicates the exposition, including the definition of throughput rate, and we do not treat it here.

to control or alter the behavior of the edge-scheduling adversary, but must come up with a protocol that works well regardless of the behavior of the ever-present (conforming) edge-scheduling adversary.

By contrast, our protocol will limit the amount of influence the node-controlling adversary has on correctness, throughput, and memory. Specifically, we will show that if a node deviates from the protocol in a sufficiently destructive manner (in a well-defined sense), then our protocol will be able to identify it as corrupted in a timely fashion. Once a corrupt node has been identified, it will be *eliminated* from the network. Namely, our protocol will call for honest nodes to refuse all communication with nodes that have been eliminated.[6] Thus, there is an inherent difference in how the two adversaries are handled: We can restrict the influence of the node-controlling adversary by eliminating the nodes it has corrupted, while the edge-scheduling adversary must be dealt with in a more ever-lasting manner.

1.1 Previous Work

To motivate the importance of the problem we consider in this paper, and to emphasize the significance of our result, it will be useful to highlight recent works in related areas. To date, routing protocols that consider adversarial networks have been of two main flavors: *End-to-End Communication* protocols that consider dynamic topologies (a notion captured by our "edge-scheduling adversary"), and *Fault Detection and Localization* protocols, which handle devious behavior of nodes (as modeled by our "node-controlling adversary").

End-to-End Communication: One of the most relevant research directions to our paper is the notion of End-to-End Communication in distributed networks, considered by Afek and Gafni [1], Awerbuch, Mansour and Shavit [8], Afek, Awebuch, Gafni, Mansour, Rosen, and Shavit [2], and Kushilevitz, Ostrovsky and Rosen [13]. Indeed, our starting point is the *Slide* protocol (also known in practical works as "gravitational flow" routing) developed in these works. It was designed to perform end-to-end communication with bounded memory in a model where (using our terminology) an edge-scheduling adversary controls the edges (subject to the constraint there is no permanent cut between the sender and receiver). The Slide protocol has proven to be incredibly useful in a variety of settings, including multi-commodity flow (Awerbuch and Leigthon [7]) and in developing routing protocols that compete well (in terms of packet loss) against an online bursty adversary ([4]). However, prior to our work there was no version of the Slide protocol that could handle malicious behavior of the nodes.

Fault Detection and Localization Protocols: At the other end, there have been a number of works that explore the possibility of a node-controlling adversary that can corrupt nodes. In particular, there is a recent line of work that considers a network consisting of a *single path* from the sender to the receiver, culminating in the recent work of Barak, Goldberg and Xiao [9] (for further

[6] The *conforming* assumption guarantees that the sender and receiver are incorruptible, and in our protocol they will identify and eliminate corrupt nodes.

background on fault localization see references therein). In this model, the adversary can corrupt any node on the path (except the sender and receiver) in a dynamic and malicious manner. Since corrupting any node on the path will sever the honest connection between S and R, the goal of a protocol in this model is *not* to guarantee that all messages sent to R are received. Instead, the goal is to *detect* faults when they occur and to *localize* the fault to a single edge.

There have been many results that provide Fault Detection (FD) and Fault Localization (FL) in this model. In Barak et. al. [9], they formalize the definitions in this model and the notion of a secure FD/FL protocol, as well as providing lower bounds in terms of communication complexity to guarantee accurate fault detection/location in the presence of a node-controlling adversary. While the Barak et. al. paper has a similar flavor to our paper, we emphasize that their protocol does not seek to guarantee successful or efficient routing between the sender and receiver. Instead, their proof of security guarantees that if a packet is deleted, malicious nodes cannot collude to convince S that no fault occurred, nor can they persuade S into believing that the fault occurred on an honest edge. Localizing the fault in their paper relies on cryptographic tools, and in particular the assumption that one-way functions exist. Although utilizing these tools (such as MACs or Signature Schemes) increases communication cost, it is shown by Goldberg, Xiao, Barak, and Redford [12] that the existence of a protocol that is able to securely detect faults (in the presence of a node-controlling adversary) implies the existence of one-way functions, and it is shown in Barak et. al. [9] that *any* protocol that is able to securely localize faults necessarily requires the intermediate nodes to have a trusted setup. The proofs of these results do not rely on the fact that there is a single path between S and R, and we can therefore extend them to the more general network encountered in our model to justify our use of cryptographic tools and a trusted setup assumption (i.e. PKI) to identify malicious behavior.

Another paper that addresses routing in the Byzantine setting is the work of Awerbuch, Holmes, Nina-Rotary and Rubens [6], though this paper does not have a fully formal treatment of security, and indeed a counter-example that challenges its security is discussed in the appendix of [9].

Error-correction in the active setting: Due to space considerations, we will not be able to give a comprehensive account of all the work in this area. Instead we highlight some of the most relevant works and point out how they differ from our setting and results. For a lengthy treatment of error-correcting codes against polynomially bounded adversaries, we refer to the work of Micali at. al [14] and references therein. It is important to note that this work deals with a graph with a single "noisy" edge, as modelled by an adversary who can partially control and modify information that crosses the edge. In particular, it does not address throughput efficiency or memory considerations in a full communication network, nor does it account for malicious behavior at the vertices. Also of relevance is the work on Rajagopalan and Schulman on error-correcting network coding [15], where they show how to correct noisy edges during distributed computation. Their work does not consider actively malicious nodes, and thus

is different from our setting. It should also be noted that their work utilizes Schulman's tree-codes [18] that allow length-flexible online error-correction. The important difference between our work and that of Schulman is that in our network setting, the amount of malicious activity of corrupt nodes is not restricted.

1.2 Our Results

To date, there has not been a protocol that has considered simultaneously a network susceptible to faults occurring due to edge-failures *and* faults occurring due to malicious activity of corrupt nodes. The end-to-end communication works are not secure when the nodes are susceptible to corruption, and the fault detection and localization works focus on a *single path* for some duration of time, and do not consider a fully distributed routing protocol that utilizes the entire network and attempts to maximize throughput efficiency while guaranteeing correctness. Indeed, our work answers one of the open questions posed in the Barak et. al. paper regarding fault localization on multiple paths. In this paper we bridge the gap between these two research areas and obtain the first routing protocol simultaneously secure against both an edge-scheduling adversary and a node-controlling adversary, even if these two adversaries attack the network using an arbitrary coordinated poly-time strategy. Furthermore, our protocol achieves comparable efficiency standards in terms of throughput and processor memory as state-of-the-art protocols that are not secure against a node-controlling adversary, and it does so using *local-control*. An informal statement of our result can be found below. We emphasize that the linear transmission rate that we achieve (assuming at least n^2 messages are sent) is asymptotically optimal, as *any* protocol operating in a network with a single path connecting sender and receiver can do no better than one packet per round.

A ROUTING THEOREM FOR ADVERSARIAL NETWORKS (Informal): If one-way functions exist, then for any n-node graph and k sufficiently large, there exists a trusted-setup *linear throughput* transmission protocol that can send n^2 messages in $O(n^2)$ rounds with $O(n^4(k+\log n))$ memory per processor that is resilient against **any** poly-time conforming Edge-Scheduling Adversary and **any** conforming poly-time Node-Controlling Adversary, with negligible (in k) probability of failure or decoding error.

	Secure Against:		Processor	Throughput Rate
	Edge-Sched?	Node-Contr?	Memory	x rounds $\rightarrow$ $f(x)$ packets
Slide Protocol of [2]	YES	NO	$O(n^2 \log n)$	$f(x) = O(x - n^2)$
Slide Protocol of [13]	YES	NO	$O(n \log n)$	$f(x) = O(x/n - n^2)$
(folklore) (Flooding + Signatures)	YES	YES	$O(1)$	$f(x) = O(x/n - n^2)$
(folklore) (Signatures + Sequence No.'s)	YES	YES	*unbounded*	$f(x) = O(x - n^2)$
Our Protocol	YES	YES	$O(n^4(k+\log n))$	$f(x) = O(x - n^2)$

Fig. 1. Comparison of Our Protocol to Related Existing Protocols and Folklore.

2 Challenges and Naïve Solutions

Before proceeding, it will be useful to consider a couple of naïve solutions that achieve the goal of *correctness* (but perform poorly in terms of *throughput*), and help to illustrate some of the technical challenges that our theorem resolves. Consider the approach of having the sender continuously *flood* a single signed packet into the network for n rounds. Since the *conforming* assumption guarantees that the network provides a path between the sender and receiver through honest nodes at every round, this packet will reach the receiver within n rounds, regardless of adversarial interference. After n rounds, the sender can begin flooding the network with the next packet, and so forth. Notice that this solution will require each processor to store and continuously broadcast a single packet at any time, and hence this solution achieves excellent efficiency in terms of *processor memory*. However, notice that the *throughput* rate is sub-linear, namely after x rounds, only $O(x/n)$ packets have been outputted by the receiver.

One idea to try to improve the throughput rate might be to have the sender streamline the process, sending packets with ever-increasing sequence numbers without waiting for n rounds to pass (or signed acknowledgments from the receiver) before sending the next packet. In particular, across each of his edges the sender will send every packet once, waiting only for the neighboring node's confirmation of receipt before sending the next packet across that edge. The protocol calls for the internal nodes to act similarly. Analysis of this approach shows that not only has the attempt to improve throughput failed (it is still $O(x/n)$ in the worst-case scenario), but additionally this modification requires arbitrarily large (polynomial in n and k) processor memory, since achieving correctness in the dynamic topology of the graph will force the nodes to remember all of the packets they see until they have broadcasted them across all adjacent edges or seen confirmation of their receipt from the receiver.

2.1 Challenges in Dealing with Node-Controlling Adversaries

In this section, we discuss some potential strategies that the node-controlling and edge-scheduling adversaries may incorporate to disrupt network communication. Although our theorem will work in the presence of *arbitrary* malicious activity of the adversarial controlled nodes (except with negligible probability), it will be instructive to list a few obvious forms of devious behavior that our protocol must protect against. It is important to stress that this list is *not* intended to be exhaustive. Indeed, we do not claim to know all the specific ways an arbitrary polynomially bounded adversary may force nodes to deviate from a given protocol, and we rigorously prove that our protocol is secure against all possible deviations.

Packet Deletion/Modification. Instead of forwarding a packet, a corrupt node "drops it to the floor" (i.e. deletes it or effectively deletes it by forever storing it in memory), or modifies the packet before passing it on. Another manifestation of this is if the sender requests fault localization information of the internal nodes,

such as providing documentation of their interactions with neighbors. A corrupt node can then block or modify information that passes through it in attempt to hide malicious activity or implicate an honest node.

Introduction of Junk/Duplicate Packets. The adversary can attempt to disrupt communication flow and "jam" the network by having corrupted nodes introduce junk packets or re-broadcast old packets. Notice that junk packets can be handled by using cryptographic signatures to prevent introduction of "new" packets, but this does not control the re-transmission of old, correctly signed packets.

Disobedience of Transfer Rules. If the protocol specifies how nodes should make decisions on where to send packets, etc., then corrupt nodes can disregard these rules, including lying to adjacent nodes about their current state.

Coordination of Edge-Failures. The edge-scheduling adversary can attempt to disrupt communication flow by scheduling edge-failures in any manner that is consistent with the *conforming* criterion. Coordinating edge-failures can be used to impede correctness, memory, and throughput in various ways: e.g. packets may become lost across a failed edge, stuck at a suddenly isolated node, or arrive at the receiver out of order. A separate issue arises concerning fault localization: When the sender requests documentation from the internal nodes, the edge-scheduling adversary can slow progress of this information, as well as attempt to protect corrupt nodes by allowing them to "play-dead" (setting all of its adjacent edges to be *down*), so that incriminating evidence cannot reach the sender.

2.2 Highlights of Our Solution

Our starting point is the Slide protocol [2], which has enjoyed practical success in networks with dynamic topologies, but is not secure against nodes that are allowed to behave maliciously. Due to space constraints, we will only highlight the main ideas of the protocol here; the interested reader can find a full exposition in [5]. We begin by viewing the edges in the graph as consisting of two directed edges, and associate to each end of a directed edge a *stack* data-structure able to hold $2n$ packets and to be maintained by the node at that end. The protocol specifies the following simple, local condition for transferring a packet across a directed edge: if there are more packets in the stack at the originating end than the terminating end, transfer a packet across the edge. Similarly, within a node's local stacks, packets are shuffled to average out the stack heights along each of its edges. Intuitively, packet movement is analogous to the flow of water: high stacks create a pressure that force packets to "flow" to neighboring lower stacks. At the source, the sender maintains the pressure by filling his outgoing stacks (as long as there is room) while the receiver relieves pressure by consuming packets and keeping his stacks empty. Loosely speaking, packets traveling to nodes "near" the sender will therefore require a very large potential, packets traveling to nodes near the receiver will require a small potential, and packet transfers near intermediate nodes will require packages to have a moderate potential. Assuming these potential requirements exist, packets will

pass from the sender with a high potential, and then "flow" downwards across nodes requiring less potential, all the way to the receiver.

Because the Slide protocol provides a fully distributed protocol that works well against an edge-scheduling adversary, our starting point was to try to extend the protocol by using digital signatures[7] to provide resilience against Byzantine attacks and arbitrary malicious behavior of corrupt nodes. This proved to be a highly nontrivial task that required us to develop a lot of additional machinery, both in terms of additional protocol ideas and novel techniques for proving correctness. We give a detailed explanation of our techniques in Section 3, but due to space considerations we have omitted the formal pseudo-code and rigorous proofs of security (these can be found in the full version, see [5]). Below we give a sample of some of the key ideas we used in ensuring our additional machinery would be provably secure against a node-controlling adversary, and yet not significantly affect throughput or memory, compared to the original Slide protocol:

Addressing the "Coordination of Edge-Scheduling" Issues. In the absence of a node-control- ling adversary, previous versions of the Slide protocol (e.g. [2]) are secure and efficient against an edge-scheduling adversary, and it will be useful to discuss how some of the challenges posed by a network with a dynamic topology are handled. First, note that the total capacity of the stack data-structure is bounded by $4n^3$. That is, each of the n nodes can hold at most $2n$ packets in each of their $2n$ stacks (along each directed edge) at any time.

- To handle the loss of packets due to an edge going down while transmitting a packet, a node is required to maintain a copy of each packet it transmits along an edge until it receives confirmation from the neighbor of successful receipt.
- To handle packets becoming stuck in some internal node's stack due to edge failures, *error-correction* is utilized to allow the receiver to decode a full message without needing every packet. In particular, if an error-correcting code allowing a fraction of λ faults is utilized, then since the capacity of the network is $4n^3$ packets, if the sender is able to pump $4n^3/\lambda$ codeword packets into the network and there is no malicious deletion or modification of packets, then the receiver will necessarily have received enough packets to decode the message.
- The Slide protocol has a natural bound in terms of memory per processor of $O(n^2 \log n)$ bits, where the bottleneck is the possibility of a node holding

[7] In this paper we use public-key operations to sign individual packets with control information. Clearly, this is too expensive to do per-packet in practice. There are methods of amortizing the cost of signatures by signing "batches" of packets; using private-key initialization [9, 12], or using a combination of private-key and public key operations, such as "on-line/off-line" signatures [10, 17]. For the sake of clarity and since the primary focus of our paper is theoretical feasibility, we restrict our attention to the straight-forward public-key setting without considering these additional cost-saving techniques.

up to $2n^2$ packets in its stacks, where each packet requires $O(\log n)$ bits to describe its position in the code.

Of course, these techniques are only valid if nodes are acting honestly, which leads us to our first extension idea.

Handling Packet Modification and Introduction of Junk Packets. Before inserting any packets into the network, the sender will authenticate each packet using his digital signature, and intermediate nodes and the receiver never accept or forward messages not appropriately signed. This simultaneously prevents honest nodes becoming bogged down with junk packets, as well as ensuring that if the receiver has obtained enough authenticated packets to decode, a node-controlling adversary cannot impede the successful decoding of the message as the integrity of the codeword packets is guaranteed by the inforgibility of the sender's signature.

Fault Detection. In the absence of a node-controlling adversary, our protocol looks almost identical to the Slide protocol of [2], with the addition of signatures that accompany all interactions between two nodes. First, the sender attempts to pump the $4n^3/\lambda$ codeword packets of the first message into the network, with packet movement exactly as in the original Slide protocol. We consider all possible outcomes:

1. The sender is able to insert all codeword packets and the receiver is able to decode. In this case, the message was transmitted successfully, and our protocol moves to transfer the next message.
2. The sender is able to insert all codeword packets, but the receiver has not received enough to decode. In this case, the receiver floods the network with a single-bit message indicating *packet deletion* has occurred.
3. The sender is able to insert all codeword packets, but the receiver cannot decode because he has received duplicated packets. Although the sender's authenticating signature guarantees the receiver will not receive junk or modified packets, a corrupt node can duplicate valid packets. Therefore, the receiver may receive enough packets to decode, but cannot because he has received duplicates. In this case, the receiver floods the network with a single message indicating the label of a duplicated packet.
4. After some amount of time, the sender still has not inserted all codeword packets. In this case, the duplication of old packets is so severe that the network has become jammed, and the sender is prevented from inserting packets even along the honest path that the conforming assumption guarantees. If the sender believes the jamming cannot be accounted for by edge-failures alone, he will halt transmission and move to localizing a corrupt node.[8] One

[8] We emphasize here the importance that the sender is able to distinguish the case that the jamming is a result of the edge-scheduling adversary's controlling of edges verses the case that a corrupt node is duplicating packets. After all, in the case of the former, there is no reward for "localizing" the fault to an edge that has failed, as *all* edges are controlled by the edge-scheduling adversary, and therefore no edge

contribution this paper makes is to prove a lower bound on the insertion rate of the sender for the Slide protocol *in the absence of the node-controlling adversary*. This bound not only alerts the sender when the jamming he is experiencing exceeds what can be expected in the absence of corrupt nodes, but it also provides a mechanism for localizing the offending node(s).

The above four cases exhaust all possibilities. Furthermore, if a transmission is not successful, the sender is not only able to *detect* the fact that malicious activity has occurred, but he is also able to distinguish the *form* (i.e. Case 2-4) of the malicious activity. Meanwhile, for the top case, our protocol enjoys (within a constant factor) an equivalent throughput rate as the original Slide protocol.

Fault Localization. Once a fault has been detected, it remains to describe how to *localize* the problem to the offending node. To this end, we use digital signatures to achieve a new mechanism we call "Routing with Responsibility." By forcing nodes to sign key parts of every communication with their neighbors during the transfer of packets, they can later be held accountable for their actions. In particular, once the sender has identified the reason for failure (Cases 2-4 above), he will request all internal nodes to return *status reports*, which are signatures on the relevant parts of the communication with their neighbors. We then prove in each case that with the complete status report from every node, the sender can identify and eliminate a corrupt node. Of course, malicious nodes may choose not to send self-incriminating information. We handle this separately as explained below.

Processor Memory. The signatures on the communication a node has with its neighbors for the purpose of fault localization is a burden on the memory required of each processor that is not encountered in the original Slide protocol. One major challenge was to reduce the amount of signed information each node must maintain as much as possible, while still guaranteeing that each node has maintained "enough" information to identify a corrupt node in the case of arbitrary malicious activity leading to a failure of type 2-4 above. The content of Theorem 32 in Section 3 demonstrates that the extra memory required of our protocol is a factor of n^2 higher than that of the original Slide protocol.

Incomplete Information. As already mentioned, we will show that as long as the sender has the complete status reports from every node, he will be able to identify a corrupt node, regardless of the reason for failure 2-4 above. However, this relies on the sender obtaining all of the relevant information; the absence of even a single node's information can prevent the localization of a fault. We address this challenge in the following ways:

1. We minimize the amount of information the sender requires of each node. This way, a node need not be connected to the sender for very many rounds in order for the sender to receive its information. Specifically, regardless of

is inherently better than another. But in the case a node is duplicating packets, if the sender can identify the node, it can eliminate it and effectively reduce the node-controlling adversary's ability to disrupt communication in the future.

the reason for failure 2-4 above, a status report consists of only n pieces of information from each node, i.e. one packet for each of its edges.

2. If the sender does not have the n pieces of information from a node, it cannot afford to wait indefinitely. After all, the edge-scheduling adversary may keep the node disconnected indefinitely, or a corrupt node may simply refuse to respond. For this purpose, we create a *blacklist* for non-responding nodes, which will disallow them from transferring codeword packets in the future. This way, anytime the receiver fails to decode a codeword as in Cases 2-4 above, the sender can request the information he needs, blacklist nodes not responding within some short amount of time, and then re-attempt to transmit the codeword using only non-blacklisted nodes. Nodes should not transfer codeword packets to blacklisted nodes, but they do still communicate with them to transfer the information the sender has requested. If a new transmission again fails, the sender will only need to request information from nodes that were participating, i.e. he will *not* need to collect new information from blacklisted nodes (although the nodes will remain blacklisted until the sender gets the original information he requested of them). Nodes will be removed from the blacklist and re-allowed to route codeword packets as soon as the sender receives their information.

THE BLACKLIST. Blacklisting nodes is a delicate matter; we want to place malicious nodes "playing-dead" on this list, while at the same time we don't want honest nodes that are temporarily disconnected from being on this list for too long. We prove in the full version (see [5]) that the occasional honest node that gets put on the blacklist won't significantly hinder packet transmission. Intuitively, this is true because any honest node that is an important link between the sender and receiver will not remain on the blacklist for very long, as his connection to the sender guarantees the sender will receive all requested information from the node in a timely manner.

Ultimately, the blacklist allows us to control the amount of malicious activity to which a single corrupt node can contribute. Indeed, we show that each failed message transmission (Cases 2-4 above) can be localized (eventually) to (at least) one corrupt node. More precisely, the blacklist allows us to argue that malicious activity can cause at most n failed transmissions before a corrupt node can necessarily be identified and eliminated. Since there are at most n corrupt nodes, this bounds the number of failed transmissions at n^2. The result of this is that other than at most n^2 failed message transmissions, our protocol enjoys the same throughput efficiency of the old Slide protocol. The formal statement of this and a sketch of the proof are the contents of Theorem 33 in Section 3.

3 Routing Against a Node-Controlling + Edge-Scheduling Adversary

3.1 Definitions

In this section, we briefly describe our protocol. Due to space constraints, a detailed presentation, including formal pseudo-code and rigorous proofs, has

been omitted (these can be found in the full version [5]). As mentioned in the Introduction, our model considers *end-to-end communication* in a network consisting of n nodes in the presence of *conforming* edge-scheduling and node-controlling adversaries. We assume a synchronous network with discrete stages, where a *stage* is defined to be the unit of time in which every edge can transfer a single packet of P bits.[9] A *round* will consist of two consecutive stages during which packets are transferred between adjacent nodes (the *Routing Phase*), followed by the *Re-Shuffle Phase* in which nodes perform (instantaneous) local maintenance of their buffers. A *transmission* (usually denoted by T) will consist of $O(n^3)$ rounds during which the sender inserts packets corresponding to a single codeword. At the end of each transmission, the receiver will broadcast an *end of transmission* message, indicating whether it could successfully decode the codeword. In the case that the receiver cannot decode, we will say that the transmission *failed*, and otherwise the transmission was *successful*.

In the case a transmission fails, the sender will determine the reason for failure (Cases 2-4 from Section 2.2, and also F2-F4 below), and request nodes to return *status reports* that correspond to a particular piece of signed communication between each node and its neighbors. We will refer to status report packets as *parcels* to clarify discussion in distinguishing them from the codeword *packets*.

The first step in providing a guarantee of efficiency (in terms of *throughput*) is to prove that every failed transmission falls under one of the following cases (the number of packets per codeword, D, will be defined in the next section):

F2. The receiver could not decode, and the sender has inserted D packets

F3. The receiver could not decode, the sender has inserted D packets, and the receiver has *not* received any duplicated packets corresponding to the current codeword

F4. The receiver could not decode and cases F2 and F3 do not happen

We describe in Section 3.3 how we identify a corrupt node in each case. The primary tool that will be used to handle case F2 will be the notion of *potential*, defined now.

Definition 31. The ***height*** H_B of any internal buffer B is the number of packets currently stored in the buffer. The ***potential*** Φ_B of the buffer is the arithmetic sum up to H_B, i.e. $\Phi_B = \sum_{i=1}^{H} i = \frac{H(H+1)}{2}$.

3.2 Description of the Node-Controlling+Edge-Scheduling Protocol

Setup. The sender has a sequence of messages $\{m_1, m_2, \dots\}$ of uniform size $M = \frac{6\sigma(P-2k)n^3}{\lambda}$ that he will expand into *codewords* $\{b_1, b_2, \dots\}$ of size $C = \frac{M}{\sigma}$ (σ is the *information rate* and λ the *error-rate* of any error-correcting code). The

[9] We assume $P > O(k + \log n)$, where k is the security parameter used for the signature scheme and n is the number of nodes. In particular, this will allow packets to carry two signatures (requires $2k$ bits) and a codeword index (requires $\log n$ bits) in addition to the codeword information.

codewords are divided into packets of size $P - 2k$ (P is the number of bits that can be transferred by an edge in a single stage, k is the security parameter), which will allow packets to have enough room to hold two signatures of size k. Since the number of packets per codeword is $D := \frac{C}{P-2k} = \frac{6(P-2k)n^3}{(P-2k)\lambda} = \frac{6n^3}{\lambda}$, if R receives $(1-\lambda)D$ distinct packets corresponding to the same codeword, he will be able to decode.

Each internal node has the following buffers:

1. *Incoming and Outgoing Buffers.* For each incoming/outgoing edge, a node will have a buffer that has the capacity to hold $2n$ packets at any given time. The receiver has one large storage buffer, and the sender has a "Copy of Current Packets" buffer to be used any time a transmission fails and needs to be repeated.
2. *Signature Buffers.* Each node has a signature buffer along each edge to keep track of incoming (resp. outgoing) information exchanged with its neighbor along that edge. The signature buffers will hold information corresponding to changes in: 1) The net number of packets passed across each adjacent edge; 2) The cumulative change in potential due to packet transfers across each adjacent edge; and 3) For each packet p, the net number of times p has passed across each adjacent edge. Each of the three items above, together with the current round index and transmission index, will be signed by the adjacent node before they are stored.
3. *Broadcast Buffer.* This is where nodes will temporarily store their neighbor's (and their own) state information that the sender will need to identify malicious activity. A node's broadcast buffer can hold the *start* and *end of transmission* parcels (see below), blacklist information, and up to n parcels of status report information for each node in the network.
4. *Data Buffer.* This keeps track of eliminated and blacklisted nodes. The sender's data buffer will also be able to store information for up to n failed transmissions, including why they failed, blacklisted nodes, and up to n status report parcels per node per failed transmission.

Also as part of the Setup, all nodes learn the relevant parameters (P, n, λ, and σ), each node receives a private key from a trusted third party for signing, and each node receives public information that allows them to verify the signature of every other node in the network.

Routing Phase. This consists of two consecutive stages during which nodes transfer codeword packets and broadcast parcels that comprise status reports and auxiliary information. The manner in which packets and parcels are transferred across a directed edge[10] $E(A, B)$ is succinctly described in the figure below. We state once and for all that if a node ever receives inaccurate or mis-signed

[10] For clarity, even though we are considering "directed edge" $E(A, B)$, we indicate communication that travels from B to A. In reality, this communication will pass across $E(B, A)$.

information, it will act as if no information was received at all (e.g. as if the edge had failed for that stage).

At the end of every transmission, the receiver will broadcast a parcel indicating if it was able to decode the previous codeword, as well as containing the label of a codeword packet he received twice (if one exists). From this, the sender will create the *start of transmission* (SOT) broadcast, which includes information concerning up to n failed transmissions, including why the transmission failed and which nodes are blacklisted (or eliminated) for those transmissions. We stress that *no node is allowed to transfer any codeword packets until it has received the complete SOT broadcast.*

Stage	A		B
1	$H_A :=$ Height of buffer along $E(A,B)$ Height of prev. p. sent (if still in A) Round prev. packet was sent Confirmation of rec. of broadcast info.	$\longrightarrow$	
		$\longleftarrow$	$H_B :=$Ht. of buffer along $E(A,B)$ Round prev. packet was received Sig's on values for edge $E(A,B)$
2	Send p. and Sig's on values for $E(A,B)$ if: • A has rec.'d SOT bdcst • B is not on A's blacklist/eliminated – $H_A > H_B$ OR – B didn't rec. prev. packet sent	$\longrightarrow$	Receive packet if: • B has rec.'d SOT bdcst • A is not on B's blacklist/elim.
		$\longleftarrow$	Broadcast Information

Fig. 2. *Description of Communication Exchange Along Directed Edge $E(A, B)$ During the Routing Phase of Some Round.*

Re-Shuffle Rules. At the end of each round, nodes will shuffle the packets they are holding to balance the distribution of packets in their incoming and outgoing buffers. After re-shuffling, all buffers will have the same number of packets, where preference will be given to outgoing buffers if perfect balancing is not possible. During the Re-Shuffle Phase, the sender will fill each of his outgoing buffers (in an arbitrary order) with packets corresponding to the current codeword. Meanwhile, the receiver will empty all of its incoming buffers into its storage buffer. If at any time R has received enough packets to decode a codeword b_i, then R outputs message m_i and empties his storage buffer.

3.3 Analysis of Our Node-Controlling + Edge-Scheduling Protocol

We state our results concerning the correctness, throughput, and memory of our adversarial routing protocol.

Theorem 32. *The memory required of each node is at most $O(n^4(k + \log n))$.*

Proof. (Sketch) Looking at the information each node is required to store in their buffers (see *Setup* of Section 3.2), the dominant expense comes from maintaining the signature buffers. The theorem follows as there are $O(n)$ such buffers, and each has the capacity to hold $D{=}O(n^3)$ packets of $P{=}O(k + \log n)$ bits.

Theorem 33. *Except for the at most n^2 transmissions that may fail due to malicious activity, our Routing Protocol enjoys linear throughput. More precisely, after x transmissions, the receiver has correctly outputted at least $x - n^2$ messages. If the number of transmissions x is quadratic in n or greater, than the failed transmissions due to adversarial behavior become asymptotically negligible. Since a transmission lasts $O(n^3)$ rounds and messages contain $O(n^3)$ bits, information is transferred through the network at a linear rate.*

We begin with a sequence of lemmas:

Lemma 1. *Every failed transmission falls under Case F2, F3, or F4; the sender (with the aide of the end of transmission parcel) can determine at the end of each transmission which case occurred.*

Proof. That Cases F2-F4 cover all possibilities is clear. The sender will know Case F2 has occurred since the sender keeps track of how many packets he has inserted in each transmission. The sender will know Case F4 has occurred if the receiver returns the label of a packet received twice (in the *end of transmission* parcel). Otherwise, a failed transmission is Case F3.

Lemma 2. *If a transmission fails and Case F4 occurred, then if the sender has collected the complete status report from every* ***participating*** *node, then the sender can identify a corrupt node.*

Proof. (Sketch) Case F4 roughly corresponds to a mixed adversarial strategy of packet deletion and packet duplication: a corrupt node has been replacing current codeword packets with duplicated packets. When a transmission T fails due to Case F4, the sender has the label of a packet p that has been received at least twice by the receiver, and a node's *status report* contains its signed communication with neighbors regarding the number of times p transferred between them. The idea is to use the status reports to find a node who *output* p more times than it *input* p. In the full version, we argue that if the sender has the complete status reports from all nodes who participated in this transmission, then he will be able to find such a node $N \in G$, and this node is necessarily corrupt.

Lemma 3. *If a transmission fails and Case F3 occurred, then if the sender has collected the complete status report from every* ***participating*** *node, then the sender can identify a corrupt node.*

Proof. (Sketch) Case F3 roughly corresponds to an adversarial strategy of packet deletion. When a transmission fails due to Case F3, a node's *status report* contains its signed communication with neighbors regarding the net number of packets transferred between them. The idea is to use the status reports to find a node who *input* more packets than it *output*. In the full version, we argue that if the sender has the complete status reports from all nodes who participated in this transmission, then he will be able to find such a node $N \in G$, and this node is necessarily corrupt.

Lemma 4. *If a transmission fails and Case F2 occurred, then if the sender has collected the complete status report from every* ***participating*** *node, then the sender can identify a corrupt node.*

Proof. (Sketch) Case F2 roughly corresponds to an adversarial strategy of packet duplication. When a transmission fails due to F2, a node's *status report* contains its signed communication with neighbors regarding the net change in potential due to the packet transfers between them.

Notice that a single packet in some internal buffer at *height* H should (if all nodes are honest) contribute this amount H to the buffer's *potential.* Since packets in the sender's buffers do not count towards potential, when a packet is *inserted* by the sender, the total potential in the network will *increase* by the height the packet assumes in the incoming buffer that receives this packet (which is at most $2n$). Since the sender inserted less than D packets in Case F2, (in the absence of malicious activity) the total potential in the network can have increased by *at most* $2nD$. Meanwhile, we argue in the full version [5] that in each of the $4D - D$ rounds in which the sender could not insert a packet, the packet movement along the active honest path for the round will necessarily cause a *decrease* of at least n in the total potential in the network. Since the maximum amount of potential added to the network (due to insertions by the sender and in the absence of malicious activity) is $2nD$, while the minimum *decrease* in potential is $3nD$, there would be a *negative* amount of potential in the network. By definition of potential, this is impossible, and thus there must be a corrupt node who is contributing to illegal increases in potential (e.g. by duplicating packets). We show in the full version [5] how the status reports (which contain information on potential changes across each edge) can be used by the sender to identify and eliminate a corrupt node.

Lemma 5. *There can be at most n failed transmissions before the sender necessarily has the complete status report from every node that participated in one of those n transmissions.*

Proof. (Sketch) A node will only be allowed to *participate* in a transmission if it is in "good standing" with the sender; i.e. the sender is not missing any status report parcel from the node. Therefore, for every failed transmission for which the sender does not have the complete status report from all *participating* nodes, there will be a distinct node $N \in G$ whose status report the sender does not have. Since there are n nodes, there are at most n such transmissions.

Proof of Theorem 33 (Sketch) We provide here only a very brief sketch of the proof, leaving the details to the full version [5]. We proceed by making a sequence of Lemmas. Theorem 33 now follows from Lemmas 1-5 as follows. There are at most n^2 failed transmissions (Cases F2-F4) since Lemma F5 states that after n failed transmissions, the sender will have the complete status report from every participating node for one of these transmissions, and then Lemmas 1-4 state that the sender can identify (and eliminate) a corrupt node. After a node has been eliminated, the network is reduced to $n - 1$ nodes, and the argument can be repeated recursively. Since there are at most n corruptible nodes, there are at most n^2 failed transmissions. Meanwhile, all successful transmissions enjoy linear throughput, as each transmission lasts $4D{=}O(n^3)$ rounds and successfully decoded codewords contain $M{=}O(n^3)$ bits.

4 Conclusion and Open Problems

In this paper, we have described a protocol that is secure simultaneously against conforming node-controlling and edge-scheduling adversaries. Our results are of a theoretical nature, with rigorous proofs of correctness and guarantees of performance. Surprisingly, our protocol shows that the additional protection against the node-controlling adversary, on top of protection against the edge-scheduling adversary, can be achieved without any additional asymptotic cost in terms of throughput.

While our results do provide a significant step in the search for protocols that work in a dynamic setting (edge-failures controlled by the edge-scheduling adversary) where some of the nodes are susceptible to corruption (by a node-controlling adversary), there remain important open questions. The original Slide protocol[11] requires each internal node to have buffers of size $O(n^2 \log n)$, while ours requires $O(n^4 \log n)$, though this can be slightly improved with additional assumptions.[12] In practice, the extra factor of n^2 may make our protocol infeasible for implementation, even for overlay networks. While the need for signatures inherently force an increase in memory per node in our protocol verses the original Slide protocol, this is not what contributes to the extra $O(n^2)$ factor. Rather, the only reason we need the extra memory is to handle the third kind of malicious behavior, which roughly corresponds to the mixed adversarial strategy of a corrupt node replacing a valid packet with an old packet that the node has duplicated. Recall that in order to detect this, for *every* packet a node sees and for every neighbor, a node must keep a (signed) record of how many times this packet has traversed the adjacent edge (the $O(n^3)$ packets per codeword and $O(n)$ neighbors per node yield the $O(n^4)$ bound on memory). Therefore, one open problem is finding a less memory-intensive way to handle this type of adversarial behavior.

Our model also makes additional assumptions that would be interesting to relax. In particular, it remains an open problem to find a protocol that provides efficient routing against a node-controlling and edge-scheduling adversary in a network that is fully *asynchronous* (without the use of timing assumptions, which can be used to replace full synchrony in our solution) and/or does not restrict the adversaries to be *conforming*. As mentioned in the Introduction, if the adversary is not conforming, then he can simply permanently disconnect the sender and receiver, disallowing any possible progress. Therefore, results in this direction would have to first define some notion of *connectedness* between sender and receiver, and then state throughput efficiency results in terms of this definition.

[11] In [13], it was shown how to modify the Slide protocol so that it only requires $O(n \log n)$ memory per internal node. We did not explore in this paper if and/or how their techniques could be applied to our protocol to similarly reduce it by a factor of n.

[12] If we are given an a-priori bound that a path-length of any conforming path is at most L, the $O(n^4 \log n)$ can be somewhat reduced to $O(Ln^3 \log n)$.

5 Acknowledgments

We thank the anonymous reviewers for their suggestions. Part of the work of the authors was done while visiting IPAM and supported in part by NSF grant 0430254. The third author was also supported in part by IBM Faculty Award, Xerox Innovation Group Award, NSF grants 0430254, 0716835, 0716389, 0830803 and U.C. MICRO grant.

References

1. Y. Afek, E. Gafni "End-to-End Communication in Unreliable Networks." *PODC, pp.* 1988.
2. Y. Afek, B. Awebuch, E. Gafni, Y. Mansour, A. Rosen, N. Shavit. "*Slide*– The Key to Poly. End-to-End Communication." *J. of Algorithms 22, pp. 158-186.* 1997.
3. Y. Afek, E. Gafni, and A. Rosén. "The Slide Mechanism With Applications In Dynamic Networks." *Proc. of the 11th ACM Symp. on PoDC, pp. 35-46.* 1992.
4. W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. "Adaptive Packet Routing For Bursty Adversarial Traffic." *J. Comput. Syst. Sci. 60(3): 482-509.* 2000.
5. Y. Amir, P. Bunn, and R. Ostrovsky. "Authenticated Adversarial Routing, Full Version." *Cornell Univ. Library arXiv, Article No. 0808.0156, http://arxiv.org/abs/0808.0156* 2008.
6. B. Awerbuch, D. Holmer, C. Nina-Rotaru, and H. Rubens. "A Secure Routing Protocol Resilient to Byzantine Failures." *WiSE, pp. 21-30.* 2002. ACM, 2002.
7. B. Awerbuch and T. Leighton. "Improved Approximation Algorithms for the Multi-Commodity Flow Problem and Local Competitive Routing in Dynamic Networks." *STOC.* 1994.
8. B. Awerbuch, Y Mansour, N Shavit "End-to-End Communication With Polynomial Overhead." *Proc. of the 30th IEEE Symp. on Foundations of Computer Science, FOCS.* 1989.
9. B. Barak, S. Goldberg, and D. Xiao. "Protocols and Lower Bounds for Failure Localization in the Internet." *27th EUROCRYPT 2008, Springer LNCS 4965, pp. 341-360.* 2008.
10. S. Even, O. Goldreich, and S. Micali. "On-Line/Off-Line Digital Signatures." *J. Cryptology 9(1): pp. 35-67.* 1996.
11. O. Goldreich. "The Foundations of Cryptography, Basic Applications." Cambridge University Press. 2004.
12. S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. "Path-Quality Monitoring in the Presence of Adversaries." *ACM SIGMETRICS Vol. 36, pp. 193-204.* June 2008.
13. E. Kushilevitz, R. Ostrovsky, and A. Rosén. "Log-Space Polynomial End-to-End Communication." *SIAM Journal of Computing 27(6): 1531-1549.* 1998.
14. S. Micali, C. Peikert, M. Sudan, and D. Wilson. "Optimal Error Correction Against Computationally Bounded Noise." *TCC LNCS 3378, pp. 1-16.* 2005.
15. S. Rajagopalan and L. Schulman "A Coding Theorem for Distributed Computation." *Proc. 26th STOC, pp. 790-799.* 1994.
16. C. E. Shannon (Jan. 1949). "Communication in the presence of noise". Proc. Institute of Radio Engineers vol. 37 (1): pp. 10-21.
17. A. Shamir and Y. Tauman. "Improved Online/Offline Signature Schemes." *CRYPTO 2001, pp. 355-367.* 2001.
18. L. Schulman. "Coding for interactive communication." *Special issue on Codes and Comp. of IEEE Transactions on Info. Theory 42(6), Part I: pp.1745-1756.* 1996.