Authenticated Adversarial Routing

Yair Amir[*] Paul Bunn[†] Rafail Ostrovsky[‡]

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 does not depend on the amount of traffic and is polynomial in the network size.) 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 make the protocol practical by reducing constants or the large (though polynomial) memory requirements per processor.

Our result is related to recent work of Barak, Goldberg and Xiao in 2008 [8] who studied fault localization in networks assuming a private-key trusted setup setting. Our work, in

[*]Johns Hopkins University Department of Computer Science. Email: yairamir@cs.jhu.edu Part of this work was done while visiting IPAM and supported in part by NSF grant 0430254.

[†]UCLA Department of Mathematics. Email: bunn@math.ucla.edu Supported in part by NSF grants 0430254, 0716835, 0716389 and 0830803.

[‡]UCLA Departments of Computer Science and Department of Mathematics. Email: rafail@cs.ucla.edu Part of this work was done while visiting IPAM and supported in part by IBM Faculty Award, Xerox Innovation Group Award, NSF grants 0430254, 0716835, 0716389, 0830803 and U.C. MICRO grant.

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 [13] 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 [14] 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 [7], 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; Error-correction; Fault Localization; Multi-parity Computation in the presence of Dishonest Majority; Communication Complexity; End-to-End Communication.

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 is a processor and each edge is a communication link. We do not assume that 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. In the protocol we present in this paper, we will not assume there is any global view of the network available to the internal nodes. Such protocols are termed "local control," in that each node can make all routing decisions based only 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 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[1]:

> Edge-Scheduling Adversary. This adversary controls the *links* between nodes every round. More precisely, at 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)[2]. 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 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

[1] 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.

[2] 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.

all communication with nodes that have been identified as corrupt[3]. Thus, there is an inherent difference in how we handle the edge-scheduling adversary verses how we handle the node-controlling adversary. We can restrict the influence of the latter by eliminating the nodes it has corrupted, while the former 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 [7], Afek, Awebuch, Gafni, Mansour, Rosen, and Shavit [2], and Kushilevitz, Ostrovsky and Rosen [12] . Indeed, our starting point is the *Slide* protocol[4] 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 that 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 [6]) 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. A comparison of various versions of the Slide protocol and our protocol is featured in Figure 1 of Section 1.2 below.

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 [8] (for further 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. [8], 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

[3]The *conforming* assumption guarantees that the sender and receiver are incorruptible, and our protocol places the responsibility of identifying and eliminating corrupt nodes on these two nodes.

[4]Also known in practical works as "gravitational flow" routing.

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 [11] 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. [8] 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 [5], 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 [8].

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 [13] 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 [14], 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 [17] 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 allowed to become corrupted by a node-controlling adversary, 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 in the presence of edge-failures and corrupt nodes. 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 does so using *local-control* protocols. An informal statement of our result and comparison of our protocol to existing protocols can be found below. Although not included in the table, 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 Edge-Sched. Ad?	Secure Against Node-Cntr. Ad?	Processor Memory	Throughput Rate x rounds$\to f(x)$ packets
Slide Protocol of [2]	*YES*	*NO*	$O(n^2 \log n)$	$f(x) = O(x - n^2)$
Slide Protocol of [12]	*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*	$O(n^{4(k+\log n)})$	$f(x) = O(x - n^2)$

Figure 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[5]. 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.

[5]An alternative approach would have the sender continue flooding the first packet, and upon receipt, the receiver floods confirmation of receipt. This alternative solution requires sequence numbers to accompany packets/confirmations, and the rule that internal nodes only keep and broadcast the packet and confirmation with largest sequence number. Although this alternative may potentially speed things up, in the worst-case it will still take $O(n)$ rounds for a single packet/confirmation pair to be transmitted.

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[6] 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 in this paper 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/receiver 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. This includes "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/receiver requests documentation from the internal nodes, the edge-scheduling adversary can slow progress of this information, as well as attempt to protect

[6]We give a formal definition of the adversary in Section 3.2.

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. We provide a detailed description of our version of the Slide protocol in Section 4, but highlight the main ideas here. 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 8 and formal pseudo-code in Section 9, as well as providing rigorous proofs of security in Section 10. However, below we first 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-controlling 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.

[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 [8, 11], or using a combination of private-key and public key operations, such as "on-line/off-line" signatures [9, 16]. 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.

- 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 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 is able to 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 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 occured, but he is also able to distinguish the *form* of the malicious activity, i.e. which case 2-4 he is in. 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 with overwhelming probability identify and eliminate a corrupt node. Of course, malicious nodes may choose not to send 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 8.2 in Section 8 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 show that regardless of the reason of failure 2-4 above, once the sender receives the status reports from every node, a corrupt node can be identified. 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

[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 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.

to receive its information. Specifically, regardless of 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 show in Theorem 8.1 and Lemma 10.9 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 a single corrupt node can contribute to. 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 fact can be found in Theorem 8.1 in Section 8, and its proof can be found in Section 10.

3 The Formal Model

It will be useful to describe two models in this section, one in the presence of an edge-scheduling adversary (all nodes act "honestly"), and one in the presence of an adversary who may "corrupt" some of the nodes in the network. In Section 4 we present an efficient protocol ("Slide") that works well in the edge-scheduling adversarial model, and we then extend this protocol in Section 8 to work in the additional presence of the node-controlling adversary.

3.1 The Edge-Scheduling Adversarial Model

We model a communication network by an undirected graph $G = (V, E)$, where $|V| = n$. Each vertex (or *node*) represents a processor that is capable of storing information (in its *buffers*) and passing information to other nodes along the edges. We distinguish two nodes, the *sender*, denoted by S, and the *receiver*, denoted by R. In our model, S has an input stream of *messages* $\{m_1, m_2, \dots\}$ of uniform size that he wishes to transmit through the network to R. As mentioned in the Introduction, the three commodities we care about are *Correctness*, *Throughput*, and *Processor Memory*.

We assume a *synchronous* network, so that there is a universal clock that each node has access to[9]. The global time is divided into discrete chunks, called *rounds*, during which nodes communicate with each other and transfer packets. Each round consists of two equal intervals of unit time called *stages*, so that all nodes are synchronized in terms of when each stage begins and ends. We assume that the edges have some fixed capacity P in terms of the amount of information that can be transmitted across them per stage. The messages will be sub-divided into packets of uniform size P, so that exactly one packet can be transferred along an edge per stage[10].

The sole purpose of the network is to transmit the messages from S to R, so S is the only node that introduces new messages into the network, and R is the only node that removes them from the network (although below we introduce a node-controlling adversary who may corrupt the intermediate nodes and attempt to disrupt the network by illegally deleting/introducing messages). Although the edges in our model are bi-directional, it will be useful to consider each link as consisting of two directed edges. Except for the *conforming* restriction (see below), we allow the edges of our network to fail and resurrect arbitrarily. We model this via an *Edge-Scheduling Adversary*, who controls the status of each edge of the network, and can alter the state of any edge at any time. We say that an edge is *active* during a given stage/round if the edge-scheduling adversary allows that edge to remain "up" for the entirety of that stage/round. We impose one restriction on the failure of edges:

Definition 3.1. An edge-scheduling adversary is *conforming* if for every round of the protocol, there exists at least one path between S and R consisting of edges that *active* for the entirety of the round.

For a given round $\mathtt{t}$, we will refer to the path guaranteed by the conforming assumption as the *active path* of round $\mathtt{t}$. Notice that although the conforming assumption guarantees the existence of an active path for each round, it is *not* assumed that any node (including S and R) is aware of what that path is. Furthermore, this path may change from one round to the next. The edge-scheduling adversary cannot affect the network in any way other than controlling the status of the edges. In the next section, we introduce a node-controlling adversary who can take control of the nodes of the network[11].

[9] Although synchronous networks are difficult to realize in practice, we can further relax the model to one in which there is a known upper-bound on the amount of time an active edge can take to transfer a packet.

[10] Our protocol for the node-controlling adversarial model will require the packets to include signatures from a cryptographic signature scheme. The security of such schemes depend on the security parameter k, and the size of the resulting signatures have size $O(k)$. Additionally, error-correction will require packets to carry with them an index of $O(\log n)$ bits. Therefore, we assume that $P \geq (k + \log n)$, so that in each time step a complete packet (with signature and index) can be transferred.

[11] The distinction between the two kinds of adversaries is made solely to emphasize the contribution of this paper.

3.2 The Node-Controlling + Edge-Scheduling Adversarial Model

This model begins with the edge-scheduling adversarial model described above, and adds a polynomially bounded Node-Controlling Adversary that is capable of corrupting *nodes* in the network. The node-controlling adversary is *malicious*, meaning that the adversary can take complete control over the nodes he corrupts, and can therefore force them to deviate from any protocol in whatever manner he likes. We further assume that the adversary is *dynamic*, which means that he can corrupt nodes at any stage of the protocol, deciding which nodes to corrupt based on what he has observed thus far[12]. For a thorough discussion of these notions, see [10] and references therein.

As in Multi-Party Computation (MPC) literature, we will need to specify an "access-structure" for the adversary.

Definition 3.2. A node-controlling adversary is *conforming* if he does not corrupt any nodes who have been or will be a part of any round's active path.

Apart from this restriction, the node-controlling adversary may corrupt whoever he likes (i.e. it is not a threshold adversary). Note that the *conforming* assumption implicitly demands that S and R are incorruptible, since they are always a part of any active path. Also, this restriction on the adversary is really more a statement about when our results remain valid. This is similar to e.g. *threshold adversary* models, where the results are only valid if the number of corrupted nodes does not exceed some threshold value t. Once corrupted, a node is forever considered to be a corrupt node that the adversary has total control over (although the adversary may choose to have the node act honestly).

Notice that because correctness, throughput, and memory are the only commodities that our model values, an *honest-but-curious* adversary is completely benign, as privacy does not need to be protected[13] (indeed, any intermediate node is presumed to be able to read any packet that is passed through it). Our techniques for preventing/detecting malicious behavior will be to incorporate a *digital signature* scheme that will serve the dual purpose of validating information that is passed between nodes, as well as holding nodes accountable for information that their signature committed them to.

We assume that there is a Public-Key Infrastructure (PKI) that allows digital signatures. In particular, before the protocol begins we choose a security parameter k sufficiently large and run a key generation algorithm for a digital signature scheme, producing $n = |G|$ (secret key, verification key) pairs (sk_N, vk_N). As output to the key generation, each processor $N \in G$ is given its own private signing key sk_N and a list of all n signature verification keys $vk_{\widehat{N}}$ for all nodes $\widehat{N} \in G$. In particular, this allows the sender and receiver to sign messages to each other that cannot be forged (except with negligible probability in the security parameter) by any other node in the system.

Edge-scheduling adversaries (as described above) are commonly used to model edge failures in networks, while the contribution of our paper is in controlling a node-controlling adversary, which has the ability to corrupt the *nodes* of the network.

[12] Although the node-controlling adversary is *dynamic*, he is still constrained by the conforming assumption. Namely, the adversary may not corrupt nodes that have been, or will be, part of any active path connecting sender and receiver.

[13] If desired, privacy can be added trivially by encrypting all packets.

4 Routing Protocol in the Edge-Scheduling Adversarial Model

In this section we formally describe our edge-scheduling protocol, which is essentially the "Slide" protocol of [2].

4.1 Definitions and High-Level Ideas

The goal of the protocol is to transmit a sequence of messages $\{m_1, m_2, \dots\}$ of uniform size from the sender S to the receiver R (refer to Section 3.1 for a complete description of the model). Each node will maintain a stack (i.e. FILO buffers) along each of its (directed) edges that can hold up to $2n$ packets concurrently. To allow for packets to become stuck in the buffers, we will utilize *error-correction* (see e.g. [10]). Specifically, the messages $\{m_1, m_2, \dots\}$ are converted into codewords $\{c_1, c_2, \dots\}$, allowing the receiver to decode a message provided he has received an appropriate number (depending on the *information rate* and *error-rate* of the code) of bits of the corresponding codeword. In this paper, we assume the existence of a error-correcting code with information rate σ and error rate λ.

As part of the setup of our protocol, we assume that the messages $\{m_1, m_2, \dots\}$ have been partitioned to have uniform size $M = \frac{6\sigma P n^3}{\lambda}$ (recall that P is the capacity of each edge and σ and λ are the parameters for the error-correction code). The messages are expanded into codewords, which will have size $C = \frac{M}{\sigma} = \frac{6Pn^3}{\lambda}$. The codewords are then divided into $\frac{C}{P} = \frac{6n^3}{\lambda}$ packets of size P. We emphasize this quantity for later use:

$$D := \frac{6n^3}{\lambda} = \textit{number of packets per codeword.} \tag{1}$$

Note that the only "noise" in our network results from undelivered packets or out-dated packets (in the edge-scheduling adversarial model, any packet that R receives has not been altered). Therefore, since each codeword consists of $D = \frac{6n^3}{\lambda}$ packets, by definition of λ, if R receives $(1-\lambda)D = (1-\lambda)\left(\frac{6n^3}{\lambda}\right)$ packets corresponding to the same codeword, he will be able to decode. We emphasize this fact:

Fact 1. If the receiver has obtained $D - 6n^3 = (1-\lambda)\left(\frac{6n^3}{\lambda}\right)$ packets from any codeword, he will be able to decode the codeword to obtain the corresponding message.

Because our model allows for edges to go up/down, we force each node to keep incoming and outgoing buffers for every *possible* edge, even if that edge isn't part of the graph at the outset. We introduce now the notion of *height* of a buffer, which will be used to determine when packets are transferred and how packets are re-shuffled between the internal buffers of a given node between rounds.

Definition 4.1. The *height* of an incoming/outgoing buffer is the number of packets currently stored in that buffer.

The presence of an edge-scheduling adversary that can force edges to fail at any time complicates the interaction between the nodes. Note that our model does not assume that the nodes are aware of the status of any of its adjacent edges, so failed edges can only be detected when information that was supposed to be passed along the edge does not arrive. We handle potential edge failures as follows. First, the incoming/outgoing buffers at either end of an edge will be given a "status" (normal or problem). Also, to account for a packet that may be lost due to edge failure during transmission across that edge, a node at the receiving end of a failed edge may have to leave room in its corresponding incoming buffer. We refer to this gap as a *ghost packet*, but emphasize that the *height* of an incoming buffer is *not* affected by ghost packets (by definition, *height* only counts packets that are present in the buffer). Similarly, when a sending node "sends" a packet across an edge, it actually only sends a copy of the packet, leaving the original packet in its outgoing buffer. We will refer to the original copy left in the outgoing buffer as a *flagged packet*, and note that flagged packets continue to contribute to the height of an outgoing buffer until they are deleted.

The codewords will be transferred sequentially, so that at any time, the sender is only inserting packets corresponding to a single codeword. We will refer to the rounds for which the sender is inserting codeword packets corresponding to the i^{th} codeword as the i^{th} *transmission.* Lemma 6.15 below states that after the sender has inserted $D - 2n^3$ packets corresponding to the same codeword, the receiver can necessarily decode. Therefore, when the sender has inserted this many packets corresponding to codeword b_i, he will clear his outgoing buffers and begin distributing packets corresponding to the next codeword b_{i+1}.

4.2 Detailed Description of the Edge-Scheduling Protocol

We describe now the two main parts of the edge-scheduling adversarial routing protocol: the Setup and the Routing Phase. For a formal presentation of the pseudo-code, see Section 5.

Setup. Each internal (i.e. not S or R) node has the following buffers:

1. *Incoming Buffers.* Recall that we view each bi-directional edge as consisting of two directed edges. Then for each incoming edge, a node will have a buffer that has the capacity to hold $2n$ packets at any given time. Additionally, each incoming buffer will be able to store a "Status" bit, the label of the "Last-Received" packet, and the "Round-Received" index (the round in which this incoming buffer last *accepted* a packet, see Definition 6.5 below). The way that this additional information is used will be described in the "Routing Rules for Receiving Node" section below.
2. *Outgoing Buffers.* For each outgoing edge, a node will have a buffer that has the capacity to hold $2n$ packets at any given time. Like incoming buffers, each outgoing buffer will also be able to store a status bit, the index label of one packet (called the "Flagged" packet), and a "Problem-Round" index (index of the most recent round in which the status bit switched to 1).

The receiver will only have incoming buffers (with capacity of one) and a large *Storage Buffer* that can hold up to D packets. Similarly, the sender has only outgoing buffers (with capacity $2n$) and the input stream of messages $\{m_1, m_2, \dots\}$ which are encoded into the codewords and divided into packets, the latter then distributed to the sender's outgoing buffers.

Also as part of the Setup, all nodes learn the relevant parameters (P, n, λ, and σ).

Routing Phase. As indicated in Section 3.1, we assume a synchronous network, so that there are well-defined rounds in which information is passed between nodes. Each round consists of two units of time, called *Stages.* The formal treatment of the Routing Phase can be found in the pseudo-code of Section 5. Informally, Figure 2 below considers a directed edge $E(A, B)$ from A (including $A = S$) to B (including $B = R$), and describes what communication each node sends in each stage.

Stage	A		B
1	$H_A :=$ Height of buffer along $E(A, B)$ Height of flagged p. (if there is one) Round prev. packet was sent	$\longrightarrow$	
		$\longleftarrow$	$H_B :=$ Height of buffer along $E(A, B)$ Round prev. packet was received
2	Send packet if: • $H_A > H_B$ *OR* • B didn't rec. prev. packet sent	$\longrightarrow$	

Figure 2: *Description of communication exchange along directed edge $E(A, B)$ during the Routing Phase of any round.*

In addition to this communication, each node must update its internal state based on the communication it receives. In particular, from the communication A receives from B in Stage 1 of any round, A can determine if B has received the most recent packet A sent. If so, A will delete this packet and switch the status of the outgoing buffer along this edge to "normal." If not, A will keep the packet as a flagged packet, and switch the status of the outgoing buffer along this edge to "problem." At the other end, if B does not receive A's Stage 1 communication *or* B does not receive a packet it was expecting from A in Stage 2, then B will leave a gap in its incoming buffer (termed a "ghost packet") and will switch this buffer's status to "problem." On the other hand, if B successfully receives a packet in Stage 2, it will switch the buffer back to "normal" status.

Re-Shuffle Rules. At the end of each round, nodes will shuffle the packets they are holding according to the following rules:

1. Take a packet from the fullest buffer and shuffle it to the emptiest buffer, provided the difference in height is at least two (respectively one) when the packet is moved between two buffers of the same type (respectively when the packet moves from an incoming buffer to an outgoing buffer). Packets will never be re-shuffled from an outgoing buffer to an incoming buffer. If two (or more) buffers are tied for having the most packets, then a packet will preferentially be chosen *from* incoming buffers over outgoing buffers (ties are broken in a round-robin fashion). Conversely, if two (or more) buffers are tied for the emptiest buffer, then a packet will preferentially be *given to* outgoing buffers over incoming buffers (again, ties are broken in a round-robin fashion).
2. Repeat the above step until the difference between the fullest buffer and the emptiest buffer does not meet the criterion outlined in Step 1.

Recall that when a packet is shuffled locally between two buffers, packets travel in a FILO manner, so that the top-most packet of one buffer is shuffled to the top spot of the next buffer. When an outgoing buffer has a flagged packet or an incoming buffer has a ghost packet, we use instead the following modifications to the above re-shuffle rules. Recall that in terms of measuring a buffer's height, flagged packets are counted but ghost packets are not.

- Outgoing buffers do not shuffle *flagged* packets. In particular, if Rule 1 above selects to transfer a packet *from* an outgoing buffer, the top-most *non-flagged* packet will be shuffled. This may mean that a gap is created between the flagged packet and the next non-flagged packet.

- Incoming buffers do not re-shuffle ghost packets. In particular, ghost packets will remain in the incoming buffer that created them, although we do allow ghost packets to slide *down* within its incoming buffer during re-shuffling. Also, packets shuffled *into* an incoming buffer are not allowed to occupy the same slot as a ghost packet (they will take the first non-occupied slot)[14].

The sender and receiver have special rules for re-shuffling packets. Namely, 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 (Fact 1 says this amount is at most $D - 6n^3$), then R outputs message m_i and deletes all packets corresponding to codeword b_i from its storage buffer that he receives in later rounds.

4.3 Analysis of the Edge-Scheduling Adversarial Protocol

We now evaluate our edge-scheduling protocol in terms of our three measurements of performance: *correctness*, *throughput*, and *processor memory*. The throughput standard expressed in Theorem 4.2 below will serve an additional purpose when we move to the node-controlling adversary setting: The sender will know that malicious activity has occurred when the throughput standard of Theorem 4.2 is not observed. Both of the theorems below will be proved rigorously in Sections 6 and 7, after presenting the pseudo-code in Section 5.

Theorem 4.2. *Each message m_i takes at most $3D$ rounds to pass from the sender to the receiver. In particular, after $O(xD)$ rounds, R will have received at least $O(x)$ messages. Since each message has size $M = \frac{6\sigma}{\lambda} P n^3 = O(n^3)$ and $D = \frac{6n^3}{\lambda} = O(n^3)$, after $O(x)$ rounds, R has received $O(x)$ bits of information, and thus our edge-scheduling adversarial protocol enjoys a linear throughput rate.*

The above theorem implicitly states that our edge-scheduling protocol is *correct*. For completeness, we also state the memory requirements of our edge-scheduling protocol, which is bottle-necked by the $O(n^2)$ packets that each internal node has the capacity to store in its buffers.

Theorem 4.3. *The edge-scheduling protocol described in Section 4.2 (and formally in the pseudo-code of Section 5) requires at most $O(n^2 \log n)$ bits of memory of the internal processors.*

[14]Note that because ghost packets do not count towards height, there appears to be a danger that the re-shuffle rules may dictate a packet gets transferred into an incoming buffer, and this packet either has no place to go (because the ghost packet occupies the top slot) or the packet increases in height (which would violate Claim 6.4 below). However, because only incoming buffers are allowed to re-shuffle packets into other incoming buffers, and the difference in height must be at least two when this happens, neither of these troublesome events can occur.

5 Pseudo-Code for the Edge-Scheduling Adversarial Protocol

Setup

DEFINITION OF VARIABLES:

$n :=$ Number of nodes in G;
$D := \frac{6n^3}{\lambda}$;
$\mathtt{T} :=$ Transmission index;
$\mathtt{t} :=$ Stage/Round index;
$P :=$ Capacity of edge (in bits);
for every $N \in G$
  for every *outgoing* **edge** $E(N,B) \in G, B \neq S$ and $N \neq R$
    $\mathsf{OUT} \in [2n] \times \{0,1\}^P$; ## Outgoing Buffer able to hold $2n$ packets
    $\tilde{p} \in \{0,1\}^P \cup \perp$; ## Copy of packet to be sent
    $sb \in \{0,1\}$; ## Status bit
    $d \in \{0,1\}$; ## Bit indicating if a packet was sent in the previous round
    $FR \in [0..6D] \cup \perp$; ## Flagged Round (index of round N first tried to send $\tilde{p}$ to B)
    $H \in [0..2n]$; ## Height of OUT. Also denoted H_{OUT} when there's ambiguity
    $H_{FP} \in [1..2n] \cup \perp$; ## Height of Flagged Packet
    $RR \in [-1..6D] \cup \perp$; ## Round Received index (from adjacent incoming buffer)
    $H_{IN} \in [0..2n] \cup \perp$; ## Height of incoming buffer of B
  for every *incoming* **edge** $E(A,N) \in G, A \neq R$ and $N \neq S$
    $\mathsf{IN} \in [2n] \times \{0,1\}^P$; ## Incoming Buffer able to hold $2n$ packets
    $p \in \{0,1\}^P \cup \perp$; ## Packet just received
    $sb \in \{0,1\}$; ## Status bit
    $RR \in [-1..6D]$; ## Round Received (index of round N last rec'd a p. from A)
    $H \in [0..2n]$; ## Height of IN. Also denoted H_{IN} when there's ambiguity
    $H_{GP} \in [1..2n] \cup \perp$; ## Height of Ghost Packet
    $H_{OUT} \in [0..2n] \cup \perp$; ## Height of outgoing buffer, or height of Flagged Packet of A
    $sb_{OUT} \in \{0,1\}$; ## Status Bit of outgoing buffer of A
    $FR \in [0..6D] \cup \perp$; ## Flagged Round index (from adjacent outgoing buffer)

INITIALIZATION OF VARIABLES:

for every $N \in G$
  for every *incoming* **edge** $E(A,N) \in G, A \neq R$ and $N \neq S$
    Initialize IN; ## Set each entry in IN to $\perp$
    $p, FR, H_{GP} = \perp$;
    $sb, sb_{OUT}, H, H_{OUT} = 0$; $RR = -1$;
  for every *outgoing* **edge** $E(N,B) \in G, B \neq S$ and $N \neq R$
    Initialize OUT; ## Set each entry in OUT to $\perp$
    $\tilde{p}, H_{FP}, RR, FR = \perp$;
    $sb, d, H, H_{IN} = 0$;

End Setup

Figure 3: Pseudo-Code for Internal Nodes' Setup for the Edge-Scheduling Adversarial Model

Sender and Receiver's Additional Setup

DEFINITION OF ADDITIONAL VARIABLES FOR SENDER:

$\mathcal{M} := \{m_1, m_2, \dots\}$ = Input Stream of Messages;
$\kappa \in [0..D]$ = Number of packets corresponding to current codeword the sender has *knowingly* inserted;

INITIALIZATION OF SENDER'S VARIABLES:

Distribute Packets; ## See Figure 6
$\kappa = 0$;

DEFINITION OF ADDITIONAL VARIABLES FOR RECEIVER:

$I_R \in [D] \times (\{0,1\}^P \cup \perp)$ = Storage Buffer to hold packets corresponding to current codeword;
$\kappa \in [0..D]$:= Number of packets received corresponding to current codeword;

INITIALIZATION OF RECEIVER'S VARIABLES:

$\kappa = 0$;
Initialize I_R; ## Sets each element of I_R to $\perp$

End Sender and Receiver's Additional Setup

Figure 4: Additional Code for Sender and Receiver Setup

Transmission T
for every $N \in G$
  for every $\mathtt{t} < 2 * (3D)$ ## The factor of 2 is for the 2 stages per round
    if $\mathtt{t} \pmod 2 = 0$ **then:** ## STAGE 1
      for every *outgoing* **edge** $E(N,B) \in G, N \neq R, B \neq S$
        if $H_{FP} = \perp$: send $(H, \perp, \perp)$; **else:** send $(H-1, H_{FP}, FR)$;
        receive (H_{IN}, RR);
        Reset Outgoing Variables;
      for every *incoming* **edge** $E(A,N) \in G, N \neq S, A \neq R$
        send (H, RR);
        $sb_{OUT} = 0$; $FR = \perp$;
        receive $(H, \perp, \perp)$ **or** (H, H_{FP}, FR); ## If $H = \perp$ or $FR > RR$, set sb_{OUT}=1; and ## H_{OUT}=H_{FP}; O.W. set H_{OUT}=H; sb_{OUT}=0;
    else if $\mathtt{t} \pmod 2 = 1$ **then:** ## STAGE 2
      for every *outgoing* **edge** $E(N,B) \in G, N \neq R, B \neq S$
        if $H_{IN} \neq \perp$ **then:** ## Received B's info.
          Create Flagged Packet;
          if (sb=1 **or** (sb=0 **and** $H_{OUT} > H_{IN}$)) **then:**
            Send Packet;
      for every *incoming* **edge** $E(A,N) \in G, N \neq S, A \neq R$
        Receive Packet;
      if $N \notin \{S, R\}$ **then:** ***Re-Shuffle***;
      else if $N = R$ **then:** ***Receiver Re-Shuffle***;
      else if $N = S$ **then:** ***Sender Re-Shuffle***;

      if $\mathtt{t} = 2(3D) - 1$ **then:** ***End of Transmission Adjustments***;
End Transmission T

Figure 5: Routing Rules I for Edge-Scheduling Adversarial Model

```
24 Reset Outgoing Variables
25   if d = 1:                                        ## N sent a packet previous round
26     d = 0;
27     if RR = ⊥ or ⊥ ≠ FR > RR                       ## Didn't receive conf. of packet receipt
28       sb = 1;
29   if RR ≠ ⊥:
30     if ⊥ ≠ FR ≤ RR:                                ## B rec'd most recently sent packet
31       if N = S then: κ = κ + 1;
32       OUT[H_FP] = ⊥; Fill Gap;                     ## Remove p̃ from OUT, shifting
                                                      ## down packets on top of p̃ if necessary
33       FR, p̃, H_FP = ⊥; sb = 0; H = H − 1;
34   if ⊥ ≠ RR < FR and ⊥ ≠ H_FP < H:                 ## B did not receive most recently sent packet
35     Elevate Flagged Packet;                        ## Swap OUT[H] and OUT[H_FP]; Set H_FP = H;

36 Create Flagged Packet
37   if sb = 0 and H > H_IN:                          ## Normal Status, will send top packet
38     p̃ = OUT[H]; H_FP = H; FR = t;

39 Send Packet
40   d = 1;
41   send (p̃, FR);

42 Receive Packet
43   receive (p, FR);
44   if H_OUT = ⊥:                                    ## Didn't Rec. A's height info.
45     sb = 1;
46     if H_GP > H or (H_GP = ⊥ and H < 2n):   H_GP = H + 1;
47   else if sb_OUT = 1 or H_OUT > H:                 ## A packet should've been sent
48     if p = ⊥:                                      ## Packet wasn't rec'd
49       sb = 1;
50       if H_GP > H or (H_GP = ⊥ and H < 2n):   H_GP = H + 1;
51     else if RR < FR:                               ## Packet was rec'd and should keep it
52       if H_GP = ⊥:   H_GP = H + 1;                 ## If no slot is saved for p, put it on top
53       sb = 0; IN[H_GP] = p; H = H + 1; H_GP = ⊥; RR = t;
54     else:                                          ## Packet was rec'd, but already had it
55       sb = 0; Fill Gap; H_GP = ⊥;                  ## See comment about Fill Gap on line 57 below
56   else:                                            ## A packet should NOT have been sent
57     sb = 0; Fill Gap; H_GP = ⊥;                    ## If packets occupied slots above the
                                                      ## Ghost Packet, then Fill Gap will Slide
                                                      ## those packets down one slot
58 End of Transmission Adjustments
59   for every outgoing edge E(N, B) ∈ G, N ≠ R, B ≠ S:
60     if H_FP ≠ ⊥:
61       OUT[H_FP] = ⊥; Fill Gap;                     ## Remove any flagged packet p̃ from OUT, shifting
                                                      ## down packets on top of p̃ if necessary
62       d, sb = 0; FR, H_FP, p̃ = ⊥; H = H − 1;
63   for every incoming edge E(A, N) ∈ G, N ≠ S, A ≠ R:
64     H_GP = ⊥; sb = 0; RR = −1; Fill Gap;
65   if N = S then: Distribute Packets;
```

Figure 6: Routing Rules for Edge-Scheduling Adversarial Model (continued)

71	***Re-Shuffle***	
72	$(M, B_F) =$ *Find Maximum Buffer*	## Node N finds its fullest buffer B_F with height M,
		## breaking ties by 1) selecting incoming buffers over
		## outgoing buffers, then 2) Round-Robin
73	$(m, B_T) =$ *Find Minimum Buffer*	## Node N finds its emptiest buffer B_T with height m,
		## breaking ties by 1) selecting outgoing buffers over
		## incoming buffers, then 2) Round-Robin
74	**if** *Packet Should Be Re-Shuffled:*	## A packet should be re-shuffled if $M - m > 1$ **or**
		## $M - m = 1$ **and** $\left\{ \begin{array}{l} B_F \text{ is an Inc. Buffer} \\ B_T \text{ is an Out. Buffer} \end{array} \right\}$
75	***Adjust Heights***	## Adjust M, m to account for Ghost, Flagged packets.
76	$SIG_{N,N} = SIG_{N,N} + (M - m - 1)$;	## Only used for (node-contr. + edge-sched.) protocol
77	***Shuffle Packet***	
78	***Re-Shuffle***	
79	***Adjust Heights***	
80	**if** B_F is an Out. Buffer **and** $H_{FP} \geq H_{OUT}$:	## H_{FP} and H_{OUT} refer to B_F's info. If true,
81	$M = M - 1$;	## then a Flagged packet is top-most non-null packet
82	**if** B_F is an Inc. Buffer **and** $\mathsf{IN}[H_{IN} + 1] \neq \perp$:	## IN and H_{IN} refer to B_F's info. If true,
83	$M = M + 1$;	## then there is a Ghost Packet creating a gap
84	**if** B_T is an Out. Buffer **and** $\mathsf{OUT}[H_{OUT}] = \perp$:	## OUT and H_{OUT} refer to B_T's info. If true,
85	$m = m - 1$;	## then there is a Flagged packet creating a gap
86	**if** B_T is an Inc. Buffer **and** $H_{GP} \neq \perp$:	## H_{GP} and H_{IN} refer to B_T's info. If true,
87	$m = m + 1$;	## then there is a Ghost Packet creating a gap
88	***Shuffle Packet***	
89	$B_T[m + 1] = B_F[M]$;	
90	$B_F[M] = \perp$;	
91	$H_{B_T} = H_{B_T} + 1$;	## H_{B_T} is the height of B_T
92	$H_{B_F} = H_{B_F} - 1$;	## H_{B_F} is the height of B_F
93	**if** B_F is an Inc. Buffer **and** $\perp \neq H_{GP} > H_{IN}$, **then:**	## H_{GP} and H_{IN} refer to B_F's info. Since B_F lost a
94	$H_{GP} = H_{IN} + 1$;	## packet, slide Ghost Packet down into top slot
95	***Sender Re-Shuffle***	
96	*Fill Packets*;	## Fills each outgoing buffer with codeword packets not
		## yet distributed, adjusting each H_{OUT} appropriately
97	***Receiver Re-Shuffle***	
98	**for every** *incoming* edge $E(A, R) \in G$:	## Reset R's Inc. Buffer to be open
99	**if** $H_{IN} > 0$:	## R rec'd a packet along this edge this round
100	**if** $IN[1]$ is a packet for current codeword:	## Also, see comments on 104 below
101	$I_R[\kappa] = IN[1]$; $\kappa = \kappa + 1$;	
102	$H_{IN} = 0$; $IN[1] = \perp$; $H_{GP} = \perp$;	
103	**if** $\kappa \geq D - 3n^3$ **then:**	## R can decode by Fact 1
104	Decode and output message;	## Also, only keep codeword packets corresponding
		## to *next* message in future rounds

Figure 7: Re-Shuffle Rules for both Edge-Scheduling *and* (Node-Controlling + Edge-Scheduling) Protocols

6 Edge-Scheduling Adversary Model: Proofs of Lemmas and Theorems

Before proving the main two theorems for the edge-scheduling adversarial protocol (Theorems 4.2 and 4.3), we will first state and prove a sequence of claims that follow immediately from the Routing and Re-Shuffle Rules of Section 4.1. We have included pseudo-code in Section 5, and when appropriate the proofs will refer to specific line numbers in the pseudo-code. In particular, we will reference a line in pseudo-code by writing (**X.YY**), where X refers to the Figure and YY to the line number. We pushed the claims and proofs that rely heavily on the pseudo-code (but are unlikely to add insight) to Section 7 so as not to distract the reader with the gory details of these proofs. Logically, these claims need to be proven first as the claims and proofs below will rely on them (even though Section 7 appears below, the proofs there do not rely on proofs here, so there is no danger of circularity).

We state and prove here the claims that will lead to Theorem 4.2 and Theorem 4.3.

Claim 6.1. *The capacity of the internal buffers of the network (not counting S or R's buffers) is* $4n(n-2)^2$.

Proof. Each node has $(n-2)$ outgoing buffers (one to each node except itself and S, **3.07**) and $(n-2)$ incoming buffers (one from each node except itself and R, **3.17**), and thus a total of $2(n-2)$ buffers. Each of these buffers has capacity $2n$ (Lemma 7.1, parts 5, 6, and 9), and there are $n-2$ internal nodes, so the internal buffer capacity of the network is $4n(n-2)^2$. ∎

Claim 6.2. *The maximum amount of potential*[15] *in the internal buffers of the network at any time is* $2n(2n+1)(n-2)^2$.

Proof. A buffer contributes the most to network potential when it is full, in which case it contributes $\sum_{i=1}^{2n} i = n(2n+1)$. Since there are $2(n-2)$ buffers per internal node, and $n-2$ internal nodes, the maximum amount of potential in the internal buffers is as claimed. ∎

We define the *height* of a packet in an incoming/outgoing buffer to be the spot it occupies in that buffer.

Claim 6.3. *After re-shuffling, (and hence at the very end/beginning of each round), all of the buffers of each node are* **balanced**. *In particular, there are no incoming buffers that have height strictly bigger than any outgoing buffers, and the difference in height between any two buffers is at most one.*

Proof of Claim 6.3. We prove this using induction (on the round index), noting that all buffers are balanced at the outset of the protocol (lines (**3.29**) and (**3.33**)). Consider any node N in the network, and assume that its buffers are all balanced at the end of some round $\mathtt{t}$. We need to show the buffers of N will remain balanced at the end of the next round $\mathtt{t}+1$. Let B_1 and B_2 denote any two buffers of N, and let h_1 be the variable denoting the height of B_1 and h_2 the height of B_2. Suppose for the sake of contradiction that $h_1 \geq h_2 + 2$ at the end of round $\mathtt{t}+1$ (after re-shuffling). Let H denote the height of the maximum buffer in N at the end of $\mathtt{t}+1$, so $H \geq h_1 \geq h_2 + 2$. Also

[15] See Definition 6.10.

let h denote the height of the minimum buffer in N at the end of $\mathtt{t}+1$, so $h \leq h_2 \leq H-2$. But then Re-Shuffle Rules dictate that N should've kept re-shuffling (**7.72-74**), a contradiction.

Similarly, assume for contradiction that there exists an incoming buffer whose height h_2 is bigger than that of some outgoing buffer that has height h_1. Let H and h be as defined above, so we have that $h \leq h_1 < h_2 \leq H$. In the case that $h_2 = H$, Re-Shuffle Rules (**7.72**) guarantee that an *incoming* buffer will be selected to take a packet from. Also, if $h = h_1$, then Re-Shuffle Rules (**7.73**) guarantee that an *outgoing* buffer will be chosen to give a packet to. Therefore, in this case a packet should have been re-shuffled (**7.74**), and hence we have contradicted the fact that we are at the end of the Re-Shuffle phase of round $\mathtt{t}$. On the other hand, if $h \neq h_1$ or $H \neq h_2$, then $H - h \geq 2$, and again Re-Shuffling should not have terminated (**7.74**). ■

The following observation is the formalization of the concept of packets "flowing downhill" that was introduced in Section 4.1.

Claim 6.4. *Every packet is inserted into one of the sender's outgoing buffers at some initial height. When (a copy of) the packet goes between any two buffers $B_1 \neq B_2$ (either across an edge or locally during re-shuffling), its height in B_2 is less than or equal to the height it had in B_1. If $B_1 = B_2$, the statement remains true EXCEPT for on line (***6.35***).*

Proof. See Section 7, where we restate and prove this in Lemma 7.11. ■

Definition 6.5. We will say that a packet is **accepted** by a buffer B in round $\mathtt{t}$ if B receives and *stores* that packet in round $\mathtt{t}$, either due to a packet transfer or re-shuffling (as on (**6.53**) or (**7.89**)).

Definition 6.6. We say that the sender **inserts** a packet into the network in round $\mathtt{t}$ if any internal node (or R) accepts the packet (as in Definition 6.5) in round $\mathtt{t}$. Note that this definition does not require that S receives the verification of receipt (i.e. that S receives the communication on (**5.06**) indicating $RR \geq FR$), so S may not be aware that a packet was inserted.

Notice that in terms of transferring packets, the above definition distinguishes between the case that a packet is accepted by B in round $\mathtt{t}$ (as defined above) and the case that a packet arrives at B in round $\mathtt{t}$ but is deleted by B (by failing the conditional statement on line (**6.51**)). As emphasized in the Introduction, *correctness* and *throughput rate* are two of the three commodities with which we will evaluate a given routing protocol. In our protocol, we will need to show that packets are not lost en route from S to R to ensure *correctness*, and meanwhile we will want to show that packets are not (overly) duplicated (since transferred packets are actually copies of the original, some packet duplication is necessary) to allow a "fast" *throughput rate*. The following two claims guarantee that packet duplication won't become problematic while simultaneously guaranteeing that packets are never deleted completely (except by R).

Claim 6.7. *Before the end of transmission* $\mathtt{T}$*, any packet that was inserted into the network during transmission* $\mathtt{T}$ *is either in some buffer (perhaps as a flagged packet) or has been received by R.*

Proof. See Section 7, where we restate and prove this in Lemma 7.12. ■

Claim 6.8. *Not counting flagged packets, there is at most one copy of any packet in the network at any time (not including packets in the sender or receiver's buffers). Looking at all copies (flagged and un-flagged) of any given packet present in the network at any time, at most one copy will* **ever** *be accepted (as in Definition 6.5) by another node.*

Proof. See Section 7, where we restate and prove this in Lemma 7.13. ∎

The following claim won't be needed until we introduce the protocol for the (Node-Controlling + Edge-Scheduling) adversarial model, but follows from the Routing Rules outlined in Section 4.1.

Claim 6.9. *At any time, an outgoing buffer has at most one flagged packet.*

Proof. See Section 7, where we restate and prove this in Corollary 7.8. ∎

The following definition formalizes the notion of "potential," and will be necessary to prove throughput performance bounds.

Definition 6.10. For any buffer[16] $B \neq S, R$ that has height h at time t, define the *potential* of B at time t, denoted by Φ_t^B, to be:

$$\Phi_t^B := \sum_{i=1}^{h} i = \frac{h(h+1)}{2}.$$

For any internal node $N \in \mathcal{P} \setminus \{R, S\}$, define the node's potential Φ_t^N to be the sum of its buffer's potentials:

$$\Phi_t^N := \sum_{\text{Buffers } B \text{ of } N} \Phi_t^B$$

Define the *network potential* Φ_t at time t to be the sum of all the internal buffers' potentials:

$$\Phi_t := \sum_{N \in \mathcal{P} \setminus \{R,S\}} \Phi_t^N$$

It will be useful to break an internal node's potential into two parts. The first part, which we will term packet duplication potential, will be the sum of the heights of the flagged packets in the node's outgoing buffers *that have already been accepted* by the neighboring node (as in Definition 6.5). Recall that a flagged packet is a packet that was sent along an outgoing edge, but the sending node is maintaining a copy of the packet until it gets confirmation of receipt. Therefore, the contribution of packet duplication potential to overall network potential is the extraneous potential; it represents the over-counting of duplicated packets. We emphasize that not all flagged packets count towards packet-duplication potential, since packets are flagged as soon as the sending node determines a packet should be sent (see line **6.38**), but the flagged packet's height does not count towards packet duplication potential until the receiving node has accepted the packet as on line (**6.53**) (which may happen in a later round or not at all). The other part of network potential will be termed non-duplicated potential, and is the sum of the heights of all non-flagged packets together with flagged packets that have not yet been accepted. Note that the separation of potential into these two parts is purely for analysis of our protocol, indeed the nodes are not able to determine if a given flagged packet contributes to *packet duplication* or *non-duplicated* potential. For convenience, we will often refer to (network) non-duplicated potential simply as (network) potential (the meaning should be clear from context).

Notice that when a node accepts a packet, its own (non-duplicated) potential instantaneously increases by the height that this packet assumes in the corresponding incoming buffer. Meanwhile, the sending node's *non-duplicated potential* drops by the height that the packet occupied in its

[16]Packets in one of the sender or receiver's buffers do not count towards potential.

outgoing buffer, and there is a simultaneous and equivalent *increase* in this sending node's *packet duplication* potential. Separating overall network potential into these two categories will be necessary to state and prove the following Lemma:

Lemma 6.11. *Every change in network potential comes from one of the following 3 events:*

1. *S inserts a packet into the network.*
2. *R receives a packet.*
3. *A packet that was sent from one internal node to another is accepted; the verification of packet receipt is received by the sending node; a packet is shuffled between buffers of the same node; or a packet is moved within a buffer.*

Furthermore, changes in network potential due to item 1) are strictly non-negative and changes due to item 2) are strictly non-positive. Also, changes in network ***non-duplicated potential*** *due to item 3) are strictly non-positive. Finally, at all times, network* ***packet duplication potential*** *is bounded between zero and $2n^3 - 8n^2 + 8n$.*

Proof. Since network potential counts the heights of the internal nodes' buffers, it only changes when these heights change, which in turn happens exclusively when there is packet movement. By reviewing the pseudo-code, we see that this happens only on lines (**6.32**), (**6.35**), (**6.53**), (**6.55**), (**6.57**), (**6.61**), (**6.64**), and (**7.89-90**). Each of these falls under one of the three items listed in the Lemma, thus proving the first statement in the Lemma. That network potential changes due to packet insertion by S are strictly non-negative is obvious (either the receiving node's potential increases by the height the packet assumed, as on (**6.53**), or the receiving node is R and the packet does not contribute to potential). Similarly, that potential change upon packet receipt by R is strictly non-positive is clear, since packets at R do not count towards potential (see Definition 6.10). Also, since only flagged packets (but not necessarily all of them) contribute to network packet duplication potential, the biggest it can be is the maximal number of flagged packets that can exist in the network at any given time, times the maximum height each flagged packet can have. By Claim 6.9, there are at most $(n-2)^2$ flagged packets in the network at any given time, and each one has maximal height $2n$ (Lemma 7.1, part 9), so network packet duplication potential is bounded by $2n^3 - 8n^2 + 8n$.

It remains to prove that changes in network non-duplicated potential due to item 3) are strictly non-positive. To do this, we look at all lines on which there is packet movement, and argue each will result in a non-positive change to non-duplicated potential. Clearly potential changes on lines (**6.32**), (**6.55**), (**6.57**), (**6.61**), and (**6.64**) are non-positive. Also, if (**6.35**) is reached, if R has already accepted the packet, then that packet's potential will count towards *duplicated* potential within the outgoing buffer, and so the change in potential as on (**6.35**) will not affect non-duplicated potential. If on the other hand R has *not* already accepted the packet, then the flagged packet still counts towards non-duplication potential in the outgoing buffer. Since the result of (**6.35**) is simply to swap the flagged packet with the top packet in the buffer, the net change in non-duplication potential is zero. That changes in potential due to re-shuffling packets (**7.89-90**) are strictly non-positive follows from Claim 6.4. It remains to check the cases that a packet that was transferred between two internal nodes is accepted (**6.53**). Notice that upon receipt there are two changes to network non-duplicated potential: it increases by the height the packet assumes in the incoming buffer it arrived at (**6.53**), and it decreases by the height the packet had in the corresponding outgoing buffer (this decrease is because the flagged packet in the outgoing buffer will count towards packet duplication potential instead of non-duplicated potential the instant the packet is accepted).

The decrease outweighs the increase since the packet's height in the incoming buffer is less than or equal to the height it had in the corresponding outgoing buffer (Claim 6.4). ■

The following Lemma will be useful in bounding the number of rounds in which no packets are inserted. We begin with the following definition:

Definition 6.12. The sender is *blocked* from inserting any packets in some round `t` if the sender is not able to insert any packets in `t` (see Definition 6.6). Let $\beta_{\mathtt{T}}$ denote the number of rounds in a transmission `T` that the sender was blocked.

Lemma 6.13. *If at any point in any transmission* `T`*, the number of blocked rounds is* $\beta_{\mathtt{T}}$*, then there has been a* ***decrease*** *in the network's non-duplicated potential by at least*[17] $n\beta_{\mathtt{T}}$.

The intuition of the proof is to argue that each blocked round creates a drop in non-duplicated potential of at least n as follows. If the sender is blocked from inserting a packet, the node N adjacent to the sender (along the active honest path) will necessarily have a full incoming buffer along its edge to the sender. By the fact that buffers are *balanced* (Lemma 6.3), this implies that all of N's outgoing buffers are also full. Meanwhile, at the opposite end of the active honest path, the node adjacent to the receiver will necessarily send a packet to the receiver if there is anything in its outgoing buffer along this edge, and this will result in a drop of potential of whatever height the packet had in the outgoing buffer. Therefore, at the start of the active honest path, the buffers are full, while at the end of the path, a packet will be transferred to height zero (in the receiver's buffer). Intuitively, it therefore seems that tracking all packet movements along the active honest path should result in a drop of potential of at least $2n$. As the counter-example in the footnote shows, this argument does not work exactly (we are only guaranteed a drop of n), but the structure of the proof is guided by this intuition. We begin with the following lemma.

Lemma 6.14. *Let $\mathcal{C} = N_1 N_2 \dots N_l$ be a path consisting of l nodes, such that $R = N_l$ and $S \notin \mathcal{C}$. Suppose that in round* `t`*, all edges $E(N_i, N_{i+1})$, $1 \le i < l$ are active for the entire round. Let ϕ denote the change in the network's non-duplicated potential caused by:*

1. *(For $1 \le i < l$) Packet transfers across $E(N_i, N_{i+1})$ in round* `t`*,*
2. *(For $1 < i < l$) Re-shuffling packets* ***into*** *N_i's outgoing buffers during* `t`*,*

Then if O_{N_1,N_2} denotes N_1's outgoing buffer along $E(N_1, N_2)$ and O denotes its height at the outset of `t`*, we have:*

- *If O_{N_1,N_2} has a flagged packet that has already been accepted by N_2* ***before*** *round* `t`*, then:*

$$\phi \le -O + l - 1 \tag{2}$$

- *Otherwise,*

$$\phi \le -O + l - 2 \tag{3}$$

[17] An initial guess that the minimal potential drop equals "$2n$" for each blocked round is incorrect. Consider the case where the active path consists of all $n-2$ intermediate nodes with the following current state: the first two nodes' buffers all have height $2n$, the next pair's buffers all have height $2n-1$, and so forth, down to the last pair of internal nodes, whose buffers all have height $n+2$. Then the drop in the network's non-duplicated potential is only $n+2$ for this round.

Proof. The proof of this lemma is rather involved and relies heavily on the pseduo-code, so we have pushed its proof to Section 7, where it is restated and proved as Lemma 7.15. ∎

We can prove Lemma 6.13 as a Corollary.

Proof of Lemma 6.13. For every blocked round $\mathtt{t}$, by the *conforming* assumption there exists a chain $\mathcal{C}_{\mathtt{t}}$ connecting the sender and receiver that satisfies the hypothesis of Lemma 6.14. Letting N_1 denote the first node on this chain (not including the sender), the fact that the round was blocked means that N_1's incoming buffer was full, and then by Lemma 6.3, so was N_1's outgoing buffer along $E(N_1, N_2)$. Since the length of the chain l is necessarily less than or equal to n, Lemma 6.14 says that the change in non-duplicated potential contributions of ϕ (see notation there) satisfy:

$$\phi \leq -O_{N_1,N_2} + l - 1 \leq -2n + n - 1 < -n \tag{4}$$

Since ϕ only records some of the changes to non-duplicated potential, we use Statement 3 of Lemma 6.11 to argue that the contributions not counted will only help the bound since they are strictly non-positive. Since we are not double counting anywhere, each blocked round will correspond to a drop in non-duplicated potential of at least $-n$, which then yields the lemma. ∎

The following Lemma will bound the number of rounds that S needs to insert packets corresponding to the same codeword.

Lemma 6.15. *If at any time $D - 2n^3$ distinct packets corresponding to some codeword b_i have been inserted into the network, then R can necessarily decode message m_i.*

Proof. Every packet that has been inserted into the network has either reached R or is in the incoming/outgoing buffer of an internal node (Claim 6.7). Since the maximum number of packets that are in the latter category is less than $4n^3$ (Claim 6.1), if $D - 2n^3$ distinct packets corresponding to b_i have been inserted, then R has necessarily received $D - 6n^3 = (1 - \lambda)\left(\frac{6n^3}{\lambda}\right)$ of these, and so by Fact 1 R can decode message m_i. ∎

We can now (restate and) prove the two main theorems of Section 4.3.

Theorem 4.2. *Each message m_i takes at most $3D$ rounds to pass from the sender to the receiver. In particular, after $O(xD)$ rounds, R will have received at least $O(x)$ messages. Since each message has size $M = \frac{6\sigma}{\lambda} P n^3 = O(n^3)$ and $D = \frac{6n^3}{\lambda} = O(n^3)$, after $O(x)$ rounds, R has received $O(x)$ bits of information, and thus our edge-scheduling adversarial protocol enjoys a linear throughput rate.*

Proof of Theorem 4.2. Let $\mathtt{t}$ denote the round that S first tries to insert packets corresponding to a new codeword b_i into the network. In each round between $\mathtt{t}$ and $\mathtt{t} + 3D$, either S is able to insert a packet or he isn't. By the pigeonhole principle, either D rounds pass in which S can insert a packet, or $2D$ rounds pass in which no packets are inserted. In the former case, R can decode by Lemma 6.15. It remains to prove the theorem in the latter case. Lemma 6.13 says that the network non-duplicated potential drops by at least n in each of the $2D$ rounds in which no packets are inserted, a total drop of $2nD$. Meanwhile, Lemma 6.11 guarantees that the *increase* to network potential between $\mathtt{t}$ and $\mathtt{t} + 3D$ caused by *duplicated potential* is at most by $2n^3 - 8n^2 + 8n$. Combining these two facts, we have that (not counting changes in potential caused by packet insertions) the network potential *drops* by at least $2nD - 2n^3 + 8n^2 - 8n$ between $\mathtt{t}$ and $\mathtt{t} + 3D$. Since network potential can never be negative, we must account for this (non-duplicated) potential drop with positive contributions to potential change. The potential already in the network at the start of `t` adds to the potential at most $4n^4-14n^3+8n^2+8n$ (Claim 6.2). Therefore, packet insertions must account for the remaining change in potential of $(2nD-2n^3+8n^2-8n)-(4n^4-14n^3+8n^2+8n) = 2nD-4n^4+(12n^3-16n) \geq 2nD-4n^4$ (where the last inequality assumes $n \geq 3$). Lemma 6.11 states that the only way network potential can increase (other than the contribution of packet duplication potential which has already been accounted for) is when S inserts a packet (a maximum increase of $2n$ per packet), so it must be that S inserted at least $(2nD-4n^4)/2n = D-2n^3$ packets into the network between `t` and $\mathtt{t}+3D$, and again R can decode by Lemma 6.15. ∎

Theorem 4.3. *The edge-scheduling protocol described in Section 4.2 (and formally in the pseudo-code of Section 5) requires at most $O(n^2 \log n)$ bits of memory of the internal processors.*

Proof of Theorem 4.3. Packets have size $\log n$ to allow the packets to be indexed. Since each internal node needs to hold at most $O(n^2)$ packets at any time (it has $2(n-2)$ buffers, each able to hold $2n$ packets), the theorem follows. ∎

7 Edge-Scheduling Protocol: Pseudo-Code Intensive Claims and Proofs

In this section we prove that our pseudo-code is consistent with the claimed properties that our protocol enjoys.

The following lemma is the first attempt to link the pseudo-code with the high-level description of what our protocol is doing. Recall that a buffer is in *normal* (respectively *problem*) status whenever its status bit sb is zero (respectively one). Also, an outgoing buffer is said to have a *flagged packet* if $H_{FP} \neq \perp$, and the flagged packet is the packet in the outgoing buffer at height H_{FP}. Notice that because the pseudo-code is written sequentially, things that conceptually happen simultaneously appear in the pseudo-code as occurring consecutively. In particular, when packets are moved between buffers, updating the buffers' contents and updating the height variables does not happen simultaneously in the code, which explains the wording of the first sentence in the following lemma.

Lemma 7.1. *At all times (i.e. all lines of code in Figures 5, 6, and 7) EXCEPT when packets travel between buffers ((**6.32-33**), (**6.52-53**), and (**7.89-90**)), along any (directed) edge $E(A,B)$ for any pair of internal nodes (A,B), we have that:*

1. *If $H_{GP} > H_{IN}$ or $H_{GP} = \perp$, then $H_{GP} = H_{IN}+1$ or $H_{GP} = \perp$ and $\mathsf{IN}[i] \neq \perp\ \forall i \in [1..H_{IN}]$ and $\mathsf{IN}[i] = \perp\ \forall i \in [H_{IN}+1..2n]$.*
2. *If $H_{GP} \leq H_{IN}$, then $\mathsf{IN}[i] \neq \perp\ \forall i \in [1..H_{GP}-1]$ and $\forall i \in [H_{GP}+1..H_{IN}+1]$, and $\mathsf{IN}[i] = \perp\ \forall i \in [H_{IN}+2..2n]$ and $\mathsf{IN}[H_{GP}] = \perp$.*
3. *If $H_{FP} > H_{OUT}$, then $sb = 1$ and $\mathsf{OUT}[i] \neq \perp\ \forall i \in [1..H_{OUT}-1]$ and $\mathsf{OUT}[H_{FP}] \neq \perp$.*
4. *If $H_{FP} = \perp$ or $H_{FP} \leq H_{OUT}$, then $\mathsf{OUT}[i] \neq \perp\ \forall i \in [1..H_{OUT}]$.*
5. *The height of IN, as defined by the number of packets (i.e. non-null entries) of IN, is equal to the value of H_{IN}.*
6. *The height of OUT, as defined by the number of packets (i.e. non-null entries) of OUT, is equal to the value of H_{OUT}.*

7. *Whenever (**6.53**) is reached,* $H_{GP} \in [1..2n]$ *and* $H_{IN} \in [0..2n-1]$.
8. *Whenever (**6.32**) is reached,* $H_{FP} \neq \perp$ *and* $H_{OUT} \in [1..2n]$.
9. *At all times (even those listed in the hypothesis above),* $H_{IN}, H_{OUT} \in [0..2n]$ *and* $H_{GP}, H_{FP} \in \perp \cup [1..2n]$ *(so the domains of these variables are well-defined).*

Additionally, during any call to Re-Shuffle:

10. *Whenever the conditional statement on line (**7.74**) is satisfied, one packet will pass between buffers. In particular, there will be a buffer that was storing the packet before the call to Re-Shuffle that will not be storing (that instance of) the packet after the reshuffle. Similarly, there will be another buffer that has filled a vacant slot with (an instance of) the packet in question.*
11. *Flagged packets do not move. More precisely, if* $H_{FP} \neq \perp$ *just before any call to Re-Shuffle, then* H_{FP} *and* $\mathsf{OUT}[H_{FP}]$ *will not change during that call to Re-Shuffle.*
12. *Either* H_{GP} *does not change during re-shuffling or* H_{GP} *has **decreased** to equal* $H_{IN}+1$. *Also, if* $H_{GP} \neq \perp$*, then* $\mathsf{IN}[H_{GP}]$ *does not get filled at any point during re-shuffling.*
13. *If* $H_{IN} < 2n$ *before Re-Shuffling, then* $H_{IN} < 2n$ *after Re-Shuffling.*

Proof of Lemma 7.1. We prove each Statement of the Lemma above simultaneously by using induction on the round and line number as follows. We first prove the Lemma holds at the outset of the protocol (base case). We then notice that the above variables only change their value in the lines excluded from the Lemma and lines (**6.35**), (**6.38**), (**6.46**), (**6.50**), (**6.55**), (**6.57**), (**6.61-62**), (**6.64**), and (**7.91-94**). In particular, we use the induction hypothesis to argue that as long as the statement of the Lemma is true going into each set of excluded lines and lines (**6.35**), (**6.38**), (**6.46**), (**6.50**), (**6.55**), (**6.57**), (**6.61-62**), (**6.64**), and (**7.91-94**), then it will remain true when the protocol leaves each of those lines. Using this technique, we now prove each Statement listed above.

Base Case. At the outset of the protocol, H_{GP} and $H_{FP} = \perp$, H_{IN} and $H_{OUT} = 0$, and all entries of IN and OUT are $\perp$ (**3.29-31** and **3.33-35**) so Statements 1-6 and 9 are true.

Induction Step. We now prove that each of the above Statements hold after leaving lines (**6.32-33**), (**6.35**), (**6.38**), (**6.46**), (**6.50**), (**6.52-53**), (**6.55**), (**6.57**), (**6.61-62**), (**6.64**), (**7.89-90**), and (**7.91-94**), provided they held upon entering these lines.

Lines (**6.32-33**). The variables in Statements 1, 2, 5, 7, do not change in these lines, and hence these Statements remain valid by the induction hypothesis. Statement 3 is vacuously true, since H_{FP} is set to $\perp$ at the end of line (**6.33**). Also, Statement 9 will remain valid as long as Statement 8 does, as H_{FP} is set to $\perp$ on line (**6.33**), and $H_{OUT} \in [0..2n]$ would follow from Statement 8 since upon entering these lines, $H_{OUT} \in [1..2n]$ (Statement 8), and so subtracting 1 from H on line (**6.33**) ensures that H_{OUT} will remain in $[0..2n-1] \subseteq [0..2n]$. The first part of Statement 8, that $H_{FP} \neq \perp$ when (**6.32**) is reached, follows immediately from Claim 7.4 below together with the fact that (**6.30**) must have been satisfied to reach (**6.32**).

We next prove Statement 6. Anytime lines (**6.32-33**) are reached, the decrease of one by H_{OUT} on (**6.33**) represents the fact that OUT should be deleting a packet on these lines. Since the induction hypothesis (applied to Statement 6) guarantees that H_{OUT} matches the number of packets (non-bottom entries) of OUT *before* lines (**6.32-33**), the changes to H_{OUT} and the height of OUT on these lines will exactly match/cancel provided OUT *does* actually decrease in height by

1 (i.e. provided $\mathsf{OUT}[H_{FP}] \neq \perp$). Since H_{FP} is changed (**6.33**) *after* deleting a packet (**6.32**), we may apply the induction hypothesis to Statements 3 and 4 to argue that $\mathsf{OUT}[H_{FP}] \neq \perp$ as long as the value of H_{FP} was not $\perp$ when line (**6.32**) was reached. This was proven above for the first part of Statement 8.

Statement 4 follows from the argument above as follows. Upon leaving (**6.33**), $H_{FP} = \perp$, so we must show $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}]$. As was argued above, $H_{FP} \neq \perp$ when (**6.32**) is reached. If $H_{FP} > H_{OUT}$ when (**6.32**) is reached, then by the induction hypothesis applied to Statement 3, on that same line $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}-1]$ and $\mathsf{OUT}[H_{FP}] \neq \perp$. The packet at height H_{FP} will be deleted on (**6.32**), so that $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}-1]$, but $\mathsf{OUT}[i] = \perp$ for all $i \geq H_{OUT}$. Then when H_{OUT} is reduced by one on (**6.33**), we will have that $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}]$, as required.

If on the other hand $H_{FP} \leq H_{OUT}$ when (**6.32**) is reached, then by the induction hypothesis applied to Statement 4, on that same line $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}]$. The packet at height H_{FP} will be deleted on (**6.32**) and the packets on top of it shifted down one if necessary, so that after (**6.32**) but before (**6.33**), we will have that $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}-1]$, but $\mathsf{OUT}[i] = \perp$ for all $i \geq H_{OUT}$. Then when H_{OUT} is reduced by one on (**6.33**), we will have that $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}]$, as required.

The second part of Statement 8 also follows from the arguments above as follows. First, it was shown in the proof of Statement 6 that $\mathsf{OUT}[H_{FP}] \neq \perp$ when (**6.32**) is reached. In particular, the height of OUT is at least one going into (**6.32**), and then the induction hypothesis applied to Statement 6 implies that $H_{OUT} \geq 1$ when (**6.32**) is reached, and the induction hypothesis applied to Statement 9 implies that $H_{OUT} \leq 2n$ when (**6.32**) is reached.

Line (**6.35**). Since only H_{FP} and OUT are modified on (**6.35**), we need only verify Statements 3, 4, 6, and 9 remain true after leaving (**6.35**). Since H_{FP} is gets the value $max(H_{OUT}, H_{FP})$ on (**6.35**), Statement 9 will be true by the induction hypothesis (applied to Statement 9). Also, the height of OUT does not change, as (**6.35**) only swaps the location of two packets already in OUT, so Statement 6 will remain true.

Statement 3 is only relevant if $H_{FP} > H_{OUT}$ before reaching (**6.35**), since otherwise $H_{FP} = H_{OUT}$ upon leaving (**6.35**), and Statement 3 will be vacuously true. On the other hand, if $H_{FP} > H_{OUT}$, then line (**6.35**) is not reached since (**6.34**) will be false.

In order to reach (**6.35**), $H_{FP} \neq \perp$ on (**6.34**), and so both H_{OUT} and H_{FP} are not equal to $\perp$ when (**6.35**) is entered (Claim 7.4), and hence $H_{FP} \neq \perp$ upon leaving (**6.35**). Also, since (**6.35**) is only reached if $H_{FP} < H_{OUT}$ (**6.34**), we use the induction hypothesis (applied to Statement 4) to argue that before reaching (**6.35**), we had that $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}]$. In particular, both $\mathsf{OUT}[H_{FP}]$ and $\mathsf{OUT}[H_{OUT}]$ are storing a packet, and the call to *Elevate Flagged Packet* simply swaps these packets, so that after the swap, it is still the case that $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}]$. Since in this case $H_{FP} = H_{OUT}$ after line (**6.35**), Statement 4 will remain true.

Line (**6.38**). H_{FP} is the only relevant value changed on (**6.38**), so it remains to prove the relevant parts of Statements 3, 4 and 9. We will show that whenever (**6.38**) is reached, $H_{OUT} \in [1..2n]$ and $\mathsf{OUT}[H_{OUT}] \neq \perp$. If we can show these two things, we will be done, since when H_{FP} is set to H_{OUT} on (**6.38**), Statement 9 will be true, Statement 4 will follow from the induction hypothesis applied to either Statement 3 or 4, and Statement 3 will not be relevant. By the induction hypothesis (applied to Statement 9), $H_{OUT} \in [0..2n]$ when (**6.38**) is reached. The fact that (**6.38**) was reached means that the conditional statement on the line before (**6.37**) was satisfied, and thus OUT is in normal status ($sb = 0$) and $H_{OUT} \in [1..2n]$. By the induction hypothesis (applied to Statement 3), the fact that $sb = 0$ going into (**6.37**) implies that $H_{FP} = \bot$ or $H_{FP} \leq H_{OUT}$ going into (**6.37**), and then the induction hypothesis (applied to Statement 4) says that $\mathsf{OUT}[H_{OUT}] \neq \bot$ when (**6.38**) is entered.

Lines (**6.46**) and (**6.50**). The parts of Statements 1, 2, and 9 involving changes to H_{GP} are the only Statements that are affected by these lines. If the conditional statement on these lines are not satisfied, then no values change, and there is nothing to prove. We therefore consider the case that the conditional statement is satisfied. Then H_{GP} is set to $H_{IN} + 1$ on these lines, and hence Statement 2 is vacuously satisfied. Since we are assuming H_{GP} changes value on (**6.46**) or (**6.50**), the conditional statement says that $H_{GP} = \bot$ or $H_{GP} > H_{IN}$ going into (**6.46**) (respectively (**6.50**)). By the induction hypothesis (applied to Statement 1), $\mathsf{IN}[i] \neq \bot$ for all $1 \leq i \leq H_{IN}$, and $\mathsf{IN}[i] = \bot$ for all $i > H_{IN}$. Therefore, since IN and H_{IN} do not change on (**6.46**) or (**6.50**), Statement 1 will remain true upon leaving these lines. Finally, for Statement 9, we need only show $H_{GP} \in [1..2n]$ upon leaving line (**6.46**) (respectively line (**6.50**)). If $H_{GP} > H_{IN}$ going into line (**6.46**) (respectively line (**6.50**)), then the change to H_{GP} is non-positive, and so the induction hypothesis applied to Statements 1 and 9 guarantee H_{GP} will be in $[1..2n]$ upon leaving these lines. On the other hand, if $H_{GP} = \bot$ going into either of these lines, then $H_{IN} < 2n$, and the induction hypothesis applied to Statement 9 indicates that $H_{IN} \in [0..2n-1]$ going into these lines, and hence $H_{GP} \in [1..2n]$ upon leaving either line.

Lines (**6.52-53**). Notice that H_{GP} necessarily equals $\bot$ when leaving (**6.53**), so Statement 2 above is vacuously satisfied. Also, neither H_{OUT}, H_{FP}, nor OUT is modified in these lines, so Statements 3, 4, 6, 8, and the parts of Statement 9 concerning these variables will remain valid by the induction hypothesis.

We prove Statement 1 first. Recall that the *height* of an incoming buffer refers to the number of (non-ghost) packets the buffer currently holds. Since H_{GP} will necessarily equal $\bot$ when leaving line (**6.53**), we must show that $\mathsf{IN}[i] \neq \bot\ \forall i \in [1..H_{IN}]$ and $\mathsf{IN}[i] = \bot\ \forall i \in [H_{IN}+1..2n]$ upon leaving line (**6.53**). Both of these follow immediately from the induction hypothesis applied to Statements 1 and 2, as follows. By the induction hypothesis applied to Statements 1, 2, and 9, either $H_{GP} = \bot$, $1 \leq H_{GP} \leq H_{IN}$, or $H_{GP} = H_{IN} + 1 \leq 2n$ when line (**6.52**) is reached. We consider each case:

- If $H_{GP} = H_{IN} + 1$ when we reach line (**6.52**), then by the induction hypothesis (applied to Statement 1) it will also be true that $\mathsf{IN}[i] \neq \bot\ \forall i \in [1..H_{IN}]$ and $\mathsf{IN}[i] = \bot\ \forall i \in [H_{IN}+1..2n]$ when this line is reached. While on line (**6.53**), first $\mathsf{IN}[H_{GP}] = \mathsf{IN}[H_{IN}+1]$ is filled with a packet, and then H_{IN} is increased by one, and so Statement 1 will remain true by the end of line (**6.53**).

- If $1 \leq H_{GP} \leq H_{IN}$ when the protocol reaches (**6.52**), then also when this line is reached we have that (by the induction hypothesis applied to Statement 2) $\mathsf{IN}[i] \neq \bot\ \forall i \in [1..H_{GP}-1]$ and $\forall i \in [H_{GP}+1..H_{IN}+1]$, and $\mathsf{IN}[i] = \bot\ \forall i \in [H_{IN}+2..2n]$ and $\mathsf{IN}[H_{GP}] = \bot$. When a packet is inserted into slot H_{GP} and H_{IN} is increased by one on line (**6.53**), we will therefore have that all slots between 1 and (the new value of) H_{IN} will have a packet, and all other slots will be $\bot$, and thus Statement 1 will hold.

- If $H_{GP} = \bot$ going into line (**6.52**), then H_{GP} will be set to $H_{IN} + 1$ on this line, and then we can repeat the argument of the top bullet point, provided $H_{IN} + 1 \leq 2n$. If $sb_{OUT} = 1$, then Statement 4 of Lemma 7.10 states that $H_{GP} \neq \perp$ when (**6.52**) is reached, contradicting the fact we are in the case $H_{GP} = \perp$. So we may assume $sb_{OUT} = 0$, and then the fact that (**6.52**) was reached means that (**6.47**) must have been satisfied because $H_{OUT} > H_{IN}$. Since both of these variables live in $[0..2n]$ by the induction hypothesis applied to Statement 9, we conclude $H_{IN} < 2n$ on (**6.47**), and it cannot change value between then and (**6.52**).

The first part of Statement 7 is proven in the above three bullet points. For the second part, if $sb_{OUT} = 0$ when (**6.47**) was evaluated earlier in the round, then the fact that (**6.53**) was reached means $H_{OUT} > H_{IN}$, and then the second part of Statement 7 follows from the induction hypothesis applied to Statement 9. If on the other hand $sb_{OUT} = 1$ when (**6.47**) was evaluated, then the second part of Statement 7 follows from Statement 5 of Lemma 7.10.

We now prove Statement 5. There are two relevant changes made on line (**6.53**) that affect Statement 5: a packet is added to $\mathsf{IN}[H_{GP}]$ and H_{IN} is increased by one. The argument in the preceding paragraph showed that when (**6.53**) is reached, $H_{GP} \in [1..2n]$ and $\mathsf{IN}[H_{GP}] = \perp$, and therefore the net effect of (**6.53**) is to increase the number of packets stored in IN by one and to increase H_{IN} by one. Therefore, since Statement 5 was true going into line (**6.53**) by the induction hypothesis, it will remain true upon leaving (**6.53**).

It remains to prove the parts of Statement 9 not yet proven, namely that at *all* times $H_{IN} \in [0..2n]$ and $H_{GP} \in \perp \cup [1..2n]$. As was proven in the third bullet point above, if (**6.52**) is satisfied, then $H_{IN} < 2n$, and hence the change there does not threaten the domain of H_{GP}. Also, (**6.53**) sets H_{GP} to $\perp$, which is again in the valid domain. Meanwhile, on (**6.53**) H_{IN} is changed to $H_{IN} + 1 \leq 2n$, where the inequality follows from the induction hypothesis applied to Statement 7.

Line (**6.55**), (**6.57**), and (**6.64**). Since IN and H_{GP} are the only relevant quantities that change value on these lines, only the relevant parts of Statements 1, 2, and 9 must be proven. Since H_{GP} is set to $\perp$ on these lines, Statement 9 is immediate and Statement 2 is vacuously true. It remains to prove Statement 1. If $H_{GP} = \perp$ going into (**6.55**), (**6.57**), or (**6.64**), then H_{GP} and IN will not change, and the inductive hypothesis (applied to Statement 1) will ensure that Statement 1 will continue to be true upon exiting any of these lines. If $1 \leq H_{GP} \leq H_{IN}$ when (**6.55**), (**6.57**), or (**6.64**) is entered, then we may apply the induction hypothesis to Statement 2 to conclude that $\mathsf{IN}[i] \neq \perp \ \forall i \in [1..H_{GP} - 1]$ and $\forall i \in [H_{GP} + 1..H_{IN} + 1]$, and $\mathsf{IN}[i] = \perp \ \forall i \in [H_{IN} + 2..2n]$ and $\mathsf{IN}[H_{GP}] = \perp$. In particular, there is a gap in IN storing a "ghost packet," and this gap will be filled when *Fill Gap* is called on (**6.55**), (**6.57**) or (**6.64**). Namely, this will shift all the packets from height $H_{GP}+1$ through $H+1$ *down* one spot, so that after *Fill Gap* is called, $\mathsf{IN}[i] \neq \perp \ \forall i \in [1..H_{IN}]$ and $\mathsf{IN}[i] = \perp \ \forall i \in [H_{IN}+1..2n]$, which is Statement 1. Finally, if $H_{GP} > H_{IN}$ when (**6.55**), (**6.57**) or (**6.64**) is entered, then *Fill Gap* will not do anything, and so IN will not change. Since Statement 1 was true going into these lines (by our induction hypothesis), it will remain true upon exiting these lines.

Line (**6.61-62**). The only relevant variables to change values on these lines are sb_{OUT}, H_{OUT}, H_{FP}, and OUT, so we need only verify Statements 3, 4, 6, and 9 remain true after leaving (**6.61-62**). First note that $H_{FP} \neq \perp$ upon reaching (**6.61**) (since (**6.60**) must be satisfied to reach (**6.61-62**)), so the induction hypothesis (applied to Statements 3 and 4) implies that $\mathsf{OUT}[H_{FP}] \neq \perp$ when (**6.61**) is reached. Therefore, $H_{OUT} \geq 1$ when (**6.61**) is reached, and hence $H_{OUT} \in [1..2n]$ upon reaching (**6.61**) by the induction hypothesis (applied to Statement 9). In particular, when H_{OUT} is reduced by one on (**6.62**), we will have that $H_{OUT} \in [0..2n-1]$ upon leaving (**6.62**), as required. Also, H_{FP} will be set to $\perp$ upon leaving (**6.62**), so Statement 9 remains true.

Statement 6 also follows from the fact that $\mathsf{OUT}[H_{FP}] \neq \perp$ when (**6.61**) is reached, as follows. Since (by induction) Statement 6 was true upon reaching (**6.61**), the packet deleted from OUT on (**6.61**) is accounted for by the drop in H_{OUT} on (**6.62**).

Statement 3 is vacuously true upon leaving (**6.62**), so it remains to prove Statement 4. This argument is identical to the one used to prove Statement 4 in lines (**6.32-33**) above.

Lines (**7.89-94**). We first prove Statements 10-13, and then address Statements 1-9. We first prove Statement 10, i.e. that before ***Shuffle Packet*** is called on (**7.77**), we have that $B_F[M] \neq \perp$ and $B_T[m+1] = \perp$.

- If B_F is an outgoing buffer and $H_{FP} = \perp$ or $H_{FP} < H_{B_F}$, then $M = H_{B_F}$ (the conditional statement on line (**7.80**) will fail), and then $B_F[M] \neq \perp$ by the induction hypothesis applied to Statement 4.
- If B_F is an outgoing buffer and $H_{FP} \geq H_{B_F}$, then $M = H_{B_F} - 1$ (the conditional statement on line (**7.80**) will pass), and then $B_F[M] \neq \perp$ by the induction hypothesis applied to Statement 3 or 4 (that $M = H_{B_F} - 1$ is greater than zero follows from the fact that $H_{FP} \neq \perp$ implies $_{FP \neq \perp}$ (Claim 7.4), and then the induction hypothesis applied to Statement 6 says $H_{B_F} > 0$).
- If B_F is an incoming buffer and $B_F[M+1] \neq \perp$, then (**7.82**) is satisfied and M is set to $M+1$ on line (**7.82**), and then by construction $B_F[M] \neq \perp$ after line (**7.83**).
- Suppose B_F is an incoming buffer and $B_F[M+1] = \perp$. Notice that the induction hypothesis applied to Statement 2 and the fact that $B_F[M+1] = \perp$ imply that $H_{GP} > H_{IN} = M$. Therefore, the induction hypothesis applied to Statement 1 implies that $B_F[M] \neq \perp$.
- If B_T is an outgoing buffer and $B_T[m] = \perp$, then the conditional statement on line (**7.84**) will be satisfied, and hence m is set to $m-1$. Thus after line (**7.85**), $B_T[m+1] = \perp$.
- If B_T is an outgoing buffer and $B_T[m] \neq \perp$, then the induction hypothesis applied to Statements 3, 4, and 6 imply that $B_T[m+1] = \perp$.
- If B_T is an incoming buffer and $H_{GP} = \perp$, then the value of m is not changed on line (**7.86**), and so $m+1 = H_{IN} + 1$. The induction hypothesis applied to Statement 1 then implies that $B_T[m+1] = \perp$.
- If B_T is an incoming buffer and $H_{GP} \neq \perp$, then $B_T[H_{IN}+2] = \perp$ by the induction hypothesis applied to Statements 1 and 2, and thus after m is changed to $m+1$ on (**7.87**), we have that $B_T[m+1] = B_T[H_{IN}+2] = \perp$, as required.

For Statements 11-13, we need to change notation slightly, since Re-Shuffling can occur between two buffers of any types (except outgoing to incoming). To prove these statements, we therefore treat 4 cases: 1) B_F is an outgoing buffer, 2) B_F is an incoming buffer, 3) B_T is an outgoing buffer, 4) B_T is an incoming buffer. We then prove the necessary Statements in each case.

Case 1. The value of $B_F[M] = \mathsf{OUT}[M]$ is changed on line (**7.90**), and hence Statement 11 will hold provided $M \neq H_{FP}$. The top two bullet points above guarantee that this is indeed the case. Statements 12 and 13 are not relevant unless B_T is an incoming buffer, which will be handled in case 4 below.

Case 2. For Statement 13, the only relevant change to H_{IN} is on line (**7.92**), where H_{IN} *decreases* in value, and hence Statement 13 will remain true. For the first part of Statement

12, the only place H_{GP} can change is line (**7.94**). But if H_{GP} does change value here, then the conditional statement on the previous line guarantees that that H_{GP} *decreases* to $H_{IN}+1$. Statement 11 and the second part of Statement 12 are not relevant to this case.

CASE 3. The value of $B_T[m+1] = \mathsf{OUT}[m+1]$ is changed on line (**7.89**), and hence Statement 11 will hold provided $m+1 \neq H_{FP}$. But we have already shown Statement 10 remains true, and in particular the slot that is filled on line (**7.89**) was vacant. If $H_{FP} \neq \perp$, then by the induction hypothesis applied to Statements 3 and 4, $\mathsf{OUT}[H_{FP}] \neq \perp$, and hence $\mathsf{OUT}[m+1] = \perp$ implies that $m+1 \neq H_{FP}$. Statements 12 and 13 are not relevant to this case.

CASE 4. Since B_T is an incoming buffer, the condition on line (**7.74**) implies that the value of m (which is the height of B_T) on line (**7.73**) must be at most $2n-2$ ($M-m>1$ and $M, m \in [0..2n]$ by induction hypothesis applied to Statement 9). Therefore, when the height of B_T is increased by one on line (**7.91**), it will be at most $2n-1$, and so Statement 13 will remain true. For the second part of Statement 12, we must show that the value of $m+1$ on line (**7.89**) is not equal to H_{GP}. In the case that $H_{GP} \neq \perp$ on line (**7.86**), the value of m will change to $H_{IN}+1$ on line (**7.87**), and then the induction hypothesis applied to Statement 1 implies that $H_{GP} \leq H_{IN}+1 = m$ and so $H_{GP} \neq m+1$ on line (**7.89**). Statement 11 and the first part of Statement 12 are not relevant for this case.

It remains to verify Statements 1-9. There are two parts to proving Statements 1 and 2; we must show they hold when B_F is an incoming buffer and also when B_T is an incoming buffer. For the latter part, Statements 1 and 2 will be true if we can show that anytime an incoming buffer's slot is filled as on line (**7.89**), the slot was either slot $H_{IN}+1$ (in the case that $H_{GP} = \perp$) or $H_{IN}+2$ (in the case that $H_{GP} \neq \perp$). These facts follow immediately from the definition of m on line (**7.73**) and lines (**7.86-87**) and (**7.89**). For the former part, Statements 1 and 2 will remain true provided the packet taken from B_F on line (**7.89**) is the top-most packet in B_F. Looking at the conditional statement on line (**7.82**), if $\mathsf{IN}[H_{IN}+1] \neq \perp$, then by the induction hypothesis applied to Statements 1 and 2, we must have that $\mathsf{IN}[H_{IN}+1]$ is the top-most non-null packet, which is the packet that will be taken from B_F on line (**7.89**) (since in this case $M = H_{IN}$ is changed to $H_{IN}+1$ on line (**7.83**)). On the other hand, if $\mathsf{IN}[H_{IN}+1] = \perp$ on line (**7.82**), then the induction hypothesis applied to statements 1 and 2 imply that $\mathsf{IN}[H_{IN}]$ is the top-most non-null packet, which is exactly the packet taken on line (**7.89**) (since the conditional statement on line (**7.82**) won't be satisfied, and hence the value of M won't be change on line (**7.83**)).

Similarly, there are two parts to proving Statements 3 and 4; we must show they hold when B_F is an outgoing buffer and also when B_T is an outgoing buffer. The former part will be true provided the packet taken from B_F on line (**7.79**) is the top-most *non-flagged* packet. If $H_{FP} = \perp$, then there is no flagged packet, and hence the packet taken from B_F should be the top packet, i.e. the packet in index $B_F[H_{OUT}]$. Investigating the definition of M on line (**7.72**) and lines (**7.80-81**) and (**7.89**) shows that this will be the case if $H_{FP} = \perp$. If $H_{FP} \neq \perp$ and $H_{FP} < H_{OUT}$, then investigating those same lines also shows the top packet will be taken from B_F (which is not flagged since $H_{FP} < H_{OUT}$ by assumption). If $H_{FP} \geq H_{OUT}$, then line (**7.80**) will be satisfied, shifting the value of M to $H_{OUT}-1$ on line (**7.81**). By the induction hypothesis applied to Statement 3, this new value of M corresponds to the top-most non-flagged packet of B_F. The latter part will be true provided the packet given to B_T takes the first free slot in B_T (in particular, the packet will not over-write a flagged packet's spot). If $B_T[H_{OUT}] \neq \perp$ on line (**7.84**), then the induction hypothesis applied to Statements 3, 4, and 6 imply that all slots of B_T between $[1..H_{OUT}]$ are non-$\perp$, and all spots above H_{OUT} are $\perp$. Therefore, (since in this case the conditional statement on line (**7.84**) fails and hence the value of m does not change on the next line) the definition of m on line (**7.73**) and line (**7.89**) show that the first free slot of B_T will be filled. On the other hand, if $B_T[H_{OUT}] = \perp$ on line (**7.84**), then by the induction hypothesis, we must have that $B_T[H_{OUT}]$ is the first free slot of B_T, and by investigating lines (**7.73**), (**7.84-85**), and (**7.89**), this is exactly the spot that is filled.

Statements 5 and 6 remain true by the fact that Statement 10 was proven true and lines (**7.91**) and (**7.92**). To satisfy the condition on line (**7.74**), it must be that $H_{B_F} = M \geq 1$ and $H_{B_T} = m < 2n$, and hence the changes made to H_{B_F} and H_{B_T} on lines (**7.91**) and (**7.92**) will guarantee the parts of Statement 9 regarding H_{OUT} and H_{IN} remain true. Also, H_{GP} remains in the appropriate demain by induction applied to Statements 9, 12, and 13. Statements 7, 8, are not relevant. ∎

Lemma 7.2. *The domains of all of the variables in Figures 3 and 4 are appropriate. In other words, the protocol never calls for more information to be stored in a node's variable (buffer, packet, etc.) than the variable has room for.*

Proof. Below we fix a node $N \in G$ and track changes to each of its variables.

Outgoing Buffers OUT (**3.08**). Each entry of OUT is initialized to $\perp$ on (**3.33**). After this point, Statement 6 of Lemma 7.1 above guarantees OUT will need to hold at most H_{OUT} packets, and since H_{OUT} is always between 0 and $2n$ (by Statement 9 of Lemma 7.1) and packets have size P, the domain for OUT is as indicated.

Copy of Packet to be Sent $\tilde{p}$ (**3.09**). This is initialized to $\perp$ on (**3.34**), and is only modified afterwards on (**6.38**), (**6.33**), and (**6.62**). By Statements 3, 4, and 9 of Lemma 7.1, $OUT[H] \neq \perp$ when $\tilde{p}$ is set on (**6.38**), and the changes on (**6.33**) and (**6.62**) reset $\tilde{p}$ to $\perp$. Therefore, the domain of $\tilde{p}$ is as indicated.

Outgoing Status Bit sb (**3.10**). This is initialized to 0 on (**3.35**), and is only modified afterwards on lines (**6.33**), (**6.28**), and (**6.62**), all of which change sb to 0 or 1, as required.

Packet Sent Bit d (**3.11**). This is initialized to 0 on (**3.35**), and is only modified afterwards on lines (**6.26**), (**6.40**), and (**6.62**), each of which change d to 0 or 1, as required.

Flagged Round Index FR (**3.12**). This is initialized to $\perp$ on (**3.34**), and is only modified afterwards on lines (**6.38**), (**6.33**), and (**6.62**). The latter two lines reset FR to $\perp$, while (**6.38**) sets FR to the index of the current stage and round $\mathtt{t}$, and since there are $3D$ rounds per transmission and 2 stages per round (**5.02**), so when FR is set to $\mathtt{t}$ on (**6.38**), it will be in $[0..6D]$, as required.

Height of Outgoing Buffer H (**3.13**). This is initialized to 0 on (**3.35**). After this point, Statement 9 of Lemma 7.1 above guarantees $H \in [0..2n]$, as required.

Height of Flagged Packet H_{FP} (**3.14**). Statement 9 of Lemma 7.1 guarantees that H_{FP} will lie in the appropriate domain at all times.

Round Adjacent Node Last Received a Packet RR (**3.15**). This is initialized to $\perp$ on (**3.34**), and is only modified afterwards when it is received on (**5.06**), where it is either set to the received value or $\perp$ if nothing was received. As discussed below, the incoming buffer's value for RR always lies in the appropriate domain domain, and hence so will the value received on (**5.06**).

Outgoing Buffer's Value for Adjacent Node's Incoming Buffer Height H_{IN} (**3.16**). This is initialized to 0 on (**3.35**), and is only modified afterwards on line (**5.06**), where it is set to the value sent on (**5.09**) by the adjacent node, or $\perp$ in case no value was received. Since the value sent on (**5.09**) will always be between 0 and $2n$ (by Statement 9 of Lemma 7.1), H_{IN} has the required domain.

Incoming Buffers IN (**3.18**). Each entry of IN is initialized to $\perp$ on (**3.29**). After this point, Statement 5 of Lemma 7.1 above guarantees IN will need to hold at most H_{IN} packets, and since H_{IN} is always between 0 and $2n$ (by Statement 9 of Lemma 7.1) and packets have size P, the domain for IN is as indicated.

Packet Just Received p (**3.19**). This is initialized to $\perp$ on (**3.30**), and is only modified afterwards on (**6.43**), where it either is set to the value sent on (**6.41**) or $\perp$ in the case no value was received. Since the value sent on (**6.41**) has the appropriate domain (i.e. the size of a packet, P), in either case p has the appropriate domain.

Incoming Status Bit sb (**3.20**). This is initialized to 0 on (**3.31**), and is only modified afterwards on lines (**6.45**), (**6.49**), (**6.53**), (**6.55**), (**6.57**), and (**6.64**), all of which change sb to 0 or 1 as required.

Round Received Index RR (**3.21**). This is initialized to -1 on (**3.31**), and is only modified afterwards on lines (**6.53**) and (**6.64**). The former sets RR to the index of the current stage and round $\mathtt{t}$, and since there are $3D$ rounds per transmission and 2 stages per round (**5.02**), setting $RR = \mathtt{t}$ as on (**6.53**) will put RR in $[0..6D]$ as required. Meanwhile, (**6.64**) resets RR to -1. Thus, at all times $RR \in \{0,1\}^{6D}$, as required.

Height of Incoming Buffer H (**3.22**). This is initialized to 0 on (**3.31**). After this point, Statement 9 of Lemma 7.1 above guarantees $H \in [0..2n]$, as required.

Height of Ghost Packet H_{GP} (**3.23**). Statement 9 of Lemma 7.1 guarantees that H_{GP} will lie in the appropriate domain at all times.

Incoming Buffer's Value for Adjacent Node's Outgoing Buffer Height H_{OUT} (**3.24**). This is initialized to 0 on (**3.31**), and is only modified afterwards on line (**5.11**), where it is set to be one of the values sent on (**5.05**) by the adjacent node, or $\perp$ in case no value was received. Since the value sent on (**5.05**) (either H_{OUT} or H_{FP}) will always be $\perp$ or a number between 1 and $2n$ (see domain argument above for an outgoing buffer's height of flagged packet variable H_{FP}), H_{OUT} has the required domain.

Incoming Buffer's Value for Adjacent Node's Status Bit sb_{OUT} (**3.25**). This is initialized to 0 on (**3.31**), and is only modified afterwards on lines (**5.10**) and (**5.11**). Both changes assign sb_{OUT} to '0' or '1', as required.

Incoming Buffer's Value for Adjacent Node's Flagged Round Index FR (**3.26**). This is initialized to $\perp$ on (**3.30**), and is only modified afterwards on lines (**5.10-11**) and (**6.43**). Each of these times, FR is either set to the value sent by the adjacent node, or $\perp$ in the case nothing was received. Since the values sent on (**5.05**) and (**6.45**) live in $[0..6D] \cup \perp$ (see argument above for an outgoing buffer's variable FR living in the appropriate domain), so does FR.

Sender's Count of Packets Inserted κ (**4.37**). We want to argue that at all times, κ corresponds to the number of packets (corresponding to the current codeword) that the sender has *knowingly* inserted. Lines (**4.39**) and (**6.68**) guarantee that $\kappa = 0$ at the outset of any transmission. The only other place κ is modified is (**6.31**) where it is incremented by one, so we must argue that (**6.31**) is reached exactly once for every packet the sender knowingly inserts. By "knowingly" inserting a packet, we means that the sender has received verification that the adjacent node has received and stored the packet, and hence the sender can delete the packet.

Suppose that in some round $\mathtt{t}$, the sender sends a packet p as on (**6.41**). By Claim 7.7 below, the sender will continue to try and send this packet to its neighbor until he receives confirmation of receipt. There are two things to show: 1) If the sender does not receive confirmation of receipt, then κ is never incremented as on (**6.31**), and 2) If the sender *does* receive confirmation of receipt, then κ is incremented *exactly once*. By "receiveing confirmation of receipt," we mean that line (**6.30**) is satisfied in some round $\mathtt{t}'$ when the sender's value for $\tilde{p}$ equals the packet p sent in round $\mathtt{t}$ (see Definition 7.6 below). Clearly, 1) will be true since (**6.31**) will never be reached if (**6.30**) is never satisfied. For 2), suppose that in some later round $\mathtt{t}' > \mathtt{t}$ the sender gets confirmation of receipt for p. Clearly line (**6.31**) is reached this round, and κ is incremented by one there. We must show κ will not be incremented due to p ever again. However, p will be deleted on line (**6.32**) of round $\mathtt{t}'$, and therefore this packet can cause the sender to reach (**6.31**) at most once. Thus, at all times κ corresponds to the number of packets (corresponding to the current codeword) that the sender has *knowingly* inserted, as desired. Since each codeword has D packets, the domain for κ is as required.

Receiver's Storage Buffer I_R (**4.40**). Each entry of I_R is initialized to $\perp$ on (**4.43**), after which it is only modified on lines (**7.101**) and (**6.66**). The latter resets I_R, while the former sets entry κ of I_R to the packet in $IN[1]$. We show below that κ will always accurately represent the number of current codeword packets the receiver has received, and hence will be a value between 0 and D. It remains to show that $IN[1]$ will always hold a packet when (**7.101**) is reached. We use Claim 7.3 below which states that for the receiver, anytime $H_{IN} > 0$, $H_{GP} = \perp$. Therefore, whenever (**7.99**) is satisfied, Statement 1 of Lemma 7.1 (together with the argument that IN has the appropriate domain) state that $IN[1]$ will hold a packet, as required.

Receiver's Number of Packets Received κ (**4.41**). We want to show that κ always equals the number of packets corresponding to the current codeword the receiver has received so far. Lines (**4.42**) and (**6.66**) guarantee that $\kappa = 0$ at the outset of any transmission. The only other place κ is modified is (**7.101**) where it is incremented by one, so we must argue that (**7.101**) is reached exactly once for every packet (corresponding to the current codeword) that the receiver receives. By Statement 1 of Lemma 7.1 and Claim 7.3 below, anytime (**7.101**) is reached, $\mathsf{IN}[1]$ necessarily stores a packet. This packet is added to I_R on (**7.101**) and then is promptly deleted from IN on (**7.102**). By Claim 6.8, the receiver will never enter (**7.100**) twice due to the same packet, and hence (**7.101**) is reached exactly once for every distinct packet corresponding to the current codeword (see comments on **7.100** and **7.104**). Therefore, κ

always equals the number of packets corresponding to the current codeword the receiver has received so far, as desired. Since there are D packets per codeword, $\kappa \in [0..D]$, as required. ■

Claim 7.3. *For any of the receiver's buffers IN, $H_{IN} = 0$ at the start of every round. Also, anytime $H_{IN} > 0$, $H_{GP} = \perp$.*

Proof. $H = H_{IN}$ is set to 0 at the outset of the protocol (**3.31**). The first statement follows immediately from line (**7.102**), where each of the receiver's incoming buffers IN have H_{IN} reset to zero during the re-shuffle phase of every round. For the second statement, we will show that whenever H changes value from 0 in any round $\mathtt{t}$, that H_{GP} will be set to $\perp$ at the same time, and neither will change value until the end of the round when H will be reset to zero during re-shuffling. In particular, the only place H can change from zero is on (**6.53**). Suppose (**6.53**) is reached in some round $\mathtt{t}$, changing H from zero to 1, and also changing H_{GP} to $\perp$. Looking at the pseudo-code, neither H nor H_{GP} can change value until line (**7.102**), where H is reset to zero. Therefore, H can only be non-zero between lines (**6.53**) and (**5.21**) (when *Receiver Re-Shuffle* is called) of a given round, and at these times H_{GP} is always equal to $\perp$. ■

Claim 7.4. *Let OUT be any outgoing buffer, and H_{FP}, FR, and sb denote the height of it flagged packet, round the packet was flagged, and status bit, respectively (see (**3.10, 3.12, 3.14**)). Then $H_{FP} = \perp \Leftrightarrow FR = \perp$. Also, anytime OUT has no flagged packets (i.e. $H_{FP} = \perp$), OUT has normal status (i.e. $sb = 0$).*

Proof. The first statement is true at the outset of the protocol (**3.34**), so it will be enough to make sure that anytime H_{FP} or FR changes value from $\perp$ to non-$\perp$ (or vice-versa), the other one also changes. Examining the pseudo-code, these changes occur only on lines (**6.33**), (**6.38**), and (**6.62**), where it is clear H_{FP} takes on a non-$\perp$ (respectively $\perp$) value if and only if FR does.

The second statement is true at the outset of the protocol (**3.34-35**). So it is enough to show: 1) anytime H_{FP} is set to $\perp$, sb is equal to zero, and 2) anytime sb changes to one, $H_{FP} \neq \perp$. The former is true since anytime H_{FP} changes to $\perp$, sb is set to zero on the same line ((**6.33**) and (**6.62**)), while the latter is true since sb only changes to one on (**6.28**), which can only be reached if $FR \neq \perp$ (**6.27**), which by the first statement of this claim implies $H_{FP} \neq \perp$. ■

Claim 7.5.

1. *Anytime sb_{OUT} is equal to 1 when Create Flagged Packet is called on line (**5.15**), $H_{FP} \neq \perp$.*
2. *Anytime Send Packet is called on line (**5.17**), the flagged packet has height at least one (i.e. H_{FP} is at least one anytime Send Packet is called).*

Proof. We prove the 2^{nd} statement by separating the proof into the following two cases.

Case 1: $sb_{OUT} = 0$ at the start of Stage 2. Since *Send Packet* is called, the conditional statement on line (**5.16**) was satisfied. Therefore, since we are in the case $sb_{OUT} = 0$ on that line, then $H_{OUT} > H_{IN}$. Tracing H_{IN} backwards, it was received on line (**5.06**) and represents the value of H_{IN} that was sent on line (**5.09**). Using the induction hypothesis applied to Statement 9 of Lemma 7.1, $H_{IN} \geq 0$ and hence the value of H_{OUT} on (**5.16**) must be at least one. Since H_{OUT} and H_{IN} cannot change between lines (**5.15**) and (**5.16**) of any round, when *Create Flagged Packet* was called, it was still true that $sb_{OUT} = 0$ and $H_{OUT} > H_{IN} \geq 0$. Therefore, line (**6.37**) will be satisfied and (**6.38**) will set $H_{FP} = H_{OUT} \geq 1$ as required.

Case 2: $sb_{OUT} = 1$ at the start of Stage 2. Let $\mathtt{t}$ denote some round where $sb_{OUT} = 1$ at the start of Stage 2. Our strategy will be to find the most recent round that sb_{OUT} switched from 0 to 1, and argue that the value that H_{FP} acquired in that round has not changed. So let $\mathtt{t}_0 + 1$ denote the most recent round that sb_{OUT} had the value 0 at any stage of the round. We argue that $sb_{OUT} = 1$ by the end of $\mathtt{t}_0 + 1$, and $sb_{OUT} = 0$ at the start of Stage 2 of round $\mathtt{t}_0$ (the round *before* $\mathtt{t}_0 + 1$) as follows:

- If sb_{OUT} equals 0 by the end of round $\mathtt{t}_0 + 1$, then it will at the start of round $\mathtt{t}_0 + 2$, contradicting the choice of $\mathtt{t}_0 + 1$.
- If $sb_{OUT} = 1$ at the start of Stage 2 of round $\mathtt{t}_0$, then sb_{OUT} must have changed its value to 0 sometime between Stage 2 of round $\mathtt{t}_0$ and the end of round $\mathtt{t}_0 + 1$ (since $sb_{OUT} = 0$ at some point of round $\mathtt{t}_0 + 1$ by definition). This can only happen on line (**6.33**) inside the *Reset Outgoing Variables* function of round $\mathtt{t}_0 + 1$ (this is the only place that sb_{OUT} can be set to zero). However, since sb_{OUT} cannot change between the time that *Reset Outgoing Variables* is called on line (**5.07**) and the end of the round, it must be that sb_{OUT} was equal to zero at the start of round $\mathtt{t}_0 + 2$, contradicting the choice of $\mathtt{t}_0 + 1$.

Now since $sb_{OUT} = 0$ at the start of round $\mathtt{t}_0 + 1$ (it cannot change between Stage 2 of $\mathtt{t}_0$ and the start of $\mathtt{t}_0 + 1$), and $sb_{OUT} = 1$ by the end of $mathttt_0 + 1$, it must have changed on line (**6.28**) of round $\mathtt{t}_0 + 1$ (this is the only line that sets sb_{OUT} to 1). In particular, the conditional statements on lines (**6.25**) and (**6.27**) must have been satisfied, and so d was equal to 1 on line (**6.25**) of round $\mathtt{t}_0 + 1$. Since d is reset to zero during Stage 1 of every round (**6.26**), it must be that d was switched from 0 to 1 on line (**6.40**) of round $\mathtt{t}_0$ (this is the only place d is set to one). Thus, we have that *Send Packet* was called on line (**5.17**) of round $\mathtt{t}_0$. We are now back in Case 1 above (but for round $\mathtt{t}_0$ instead of $\mathtt{t}$), and thus H_{FP} was set to a value of at least 1 on line (**6.38**) of round $\mathtt{t}_0$. It remains to argue that H_{FP} does not decrease in value between round $\mathtt{t}_0$ and line (**5.17**) of round $\mathtt{t}$. But H_{FP} can only change value on lines (**6.33**), (**6.35**), and (**6.38**). For round $\mathtt{t}_0$, the former two of these lines have both passed when the latter is called (setting $H_{FP} \geq 1$ as in Case 1). Meanwhile, between $\mathtt{t}_0 + 1$ and $\mathtt{t}$, we know that (**6.33**) and (**6.38**) cannot be reached, as this would imply the value of sb_{OUT} is zero sometime after $\mathtt{t}_0 + 1$, contradicting the choice of $\mathtt{t}_0 + 1$. The only other place H_{FP} can change is (**6.35**), which can only *increase* H_{FP}. Thus in any case, $\perp \neq H_{FP} \geq 1$ when *Send Packet* is called on (**5.17**) of round $\mathtt{t}$.

The proof of the 1^{st} statement follows from the proof given in Case 2 above. ∎

Definition 7.6. We will say that an outgoing buffer gets ***confirmation of receipt*** for a packet p that it sent across its adjacent edge whenever line (**6.30**) (alternatively line (**12.46**) for the node-controlling + edge-controlling protocol of Sections 8-11) is reached and satisfied and the packet subsequently deleted on (**6.32**) (respectively (**12.50**)) is (a copy of) p.

Claim 7.7. *Suppose (an instance of) a packet p is accepted by node B in round $\mathtt{t}$ (using the definition of "accepted" from Definition 6.5). Then:*

1. *Let $\mathtt{t}'$ be the first round after*[18] *$\mathtt{t}$ in which B attempts to send (a copy of) this packet across any outgoing edge. Then the corresponding outgoing buffer* OUT *of B will necessarily have normal status at the start of Stage 2 of $\mathtt{t}'$.*

[18] The Claim remains valid even if $\mathtt{t}'$ is a round in a *different* transmission than $\mathtt{t}$.

2. *If B fails to get confirmation of receipt for the packet in the following round (i.e. either RR is not received on (**5.06**) of round $\mathtt{t}'+1$, or it is received but $RR < FR$), then* OUT *enters problem status as on (**6.28**) of round $\mathtt{t}'+1$.* OUT *will remain in problem status until the end of the transmission or until the round in which it gets confirmation of receipt (i.e. until RR is received as on (**5.06**) with $RR \geq \mathtt{t}'$).*

3. *From the time p is first flagged as on (**6.38**) of round $\mathtt{t}'$ through the time B does get confirmation of receipt (or through the end of the transmission, whichever comes first), B will not have any other flagged packets, i.e. $\tilde{p}, \mathsf{OUT}[H_{FP}] = p$ and $FR = \mathtt{t}'$.*

Proof. We prove Statement 1 by contradiction. Let $\mathtt{t}'$ denote the first round after $\mathtt{t}$ in which B attempts to send (a copy of) p across an edge $E(B,C)$, i.e. $\mathtt{t}'$ is the first round after $\mathtt{t}$ that *Send Packet* is called by B's outgoing buffer OUT such that the $\tilde{p}$ that appears on line (**6.38**) of that round corresponds to p. For the sake of contradiction, assume that $sb_{OUT} = 1$ at the start of Stage 2 of round $\mathtt{t}'$. Since sb_{OUT} cannot change between the start of Stage 2 and the time that *Create Flagged Packet* is called on line (**5.15**), we must have that $sb_{OUT} = 1$ on line (**6.37**) of round $\mathtt{t}'$, and hence (**6.38**) is not reached that round. In particular, when *Send Packet* is called on line (**5.17**) (as it must be by the fact that p was sent during round $\mathtt{t}'$), the packet $\tilde{p}$ that is sent (which is p) was set in some previous round. Let $\widetilde{\mathtt{t}}$ denote the most recent round for which $\tilde{p}$ was set to p as on (**6.38**) (this is the only line which sets $\tilde{p}$). Then by assumption $\widetilde{\mathtt{t}} < \mathtt{t}'$, and OUT had normal status at the start of Stage 2 of round $\widetilde{\mathtt{t}}$ (in order for (**6.38**) to be reached). Since OUT had normal status at the start of Stage 2 of round $\widetilde{\mathtt{t}}$, but by assumption OUT had problem status at the start of Stage 2 of round $\mathtt{t}'$, let $\widehat{\mathtt{t}}$ denote the first round such that $\widetilde{\mathtt{t}} < \widehat{\mathtt{t}} \leq \mathtt{t}'$ and such that OUT had problem status at the start of Stage 2 of $\widehat{\mathtt{t}}$. Since the only place OUT switches status from normal to problem is on (**6.28**), this line must have been reached in round $\widehat{\mathtt{t}}$. In particular, this implies that (**6.25**) was satisfied in round $\widehat{\mathtt{t}}$, which in turn implies that *Send Packet* was called in round $\widehat{\mathtt{t}}-1$ (since d is re-set to zero at the end of Stage 1 of every round as on (**6.26**)). But this is a contradiction, since $\widetilde{\mathtt{t}} \leq \widehat{\mathtt{t}}-1 < \mathtt{t}'$, and so $p = \tilde{p}$ was sent in a round before $\mathtt{t}'$, contradicting the choice of $\mathtt{t}'$.

For Statement 2, since B sent p in round $\mathtt{t}'$ and OUT had normal status at the Start of Stage 2 of this round, we have that $H_{OUT} > H_{IN}$ on line (**5.16**) (so that *Send Packet* could be called). Since sb_{OUT}, H_{OUT}, and H_{IN} cannot change between (**5.15**) and (**5.17**) of any round, FR is set to $\mathtt{t}'$ on (**6.38**) of round $\mathtt{t}'$. Also, $d = 1$ after the call to *Send Packet* of round $\mathtt{t}'$ (**6.40**). Notice that neither FR nor d can change value between the call to *Create Flagged Packet* in round $\mathtt{t}'$ and the call to *Reset Outgoing Variables* in the following round. Therefore, if B does not receive RR or if $RR < FR = \mathtt{t}'$ when *Reset Outgoing Variables* is called in round $\mathtt{t}'+1$, then (**6.25**) and (**6.27**) will be satisfied, and hence OUT will enter problem status in round $\mathtt{t}'+1$. That OUT remains in problem status until the end of the transmission or until the round in which RR is received on (**5.06**) with $RR \geq \mathtt{t}'$ now follows from the following subclaim. (Warning: the following subclaim switches notation. In particular, to apply the subclaim here, replace $(\mathtt{t}, \mathtt{t}_0)$ of the subclaim with $(\mathtt{t}'+1, \mathtt{t}')$.)

Subclaim. Suppose that at the start of Stage 2 of some round $\mathtt{t}$, an outgoing buffer OUT has problem status and $\perp \neq FR = \mathtt{t}_0$. Then OUT will remain in problem status until the end of the transmission or until the round in which RR is received on (**5.06**) with $RR \geq \mathtt{t}_0$.

Proof. OUT will certainly return to normal status by the end of the transmission (**6.62**), in which case there is nothing to show. So suppose that $\mathtt{t}' > \mathtt{t}$ is such that OUT first returns to normal status (in the same transmission as $\mathtt{t}$) as on (**6.33**) of round $\mathtt{t}'$. In particular, lines (**6.29**) and (**6.30**) were both satisfied, so OUT must have received RR on (**5.06**) earlier in round $\mathtt{t}'$, with $RR \geq FR$. If the value of FR on line (**6.30**) equals $\mathtt{t}_0$, then the proof is complete. So we show by contradiction that this must be the case.

Assume for the sake of contradiction that $FR \neq \mathtt{t}_0$ on line (**6.30**) of round $\mathtt{t}'$. Since FR was equal to $\mathtt{t}_0$ at the start of Stage 2 of round $\mathtt{t}$ by hypothesis, FR must have changed at some point between Stage 2 of round $\mathtt{t}$ and round $\mathtt{t}'$. Notice that between these rounds, FR can only change values on lines (**6.33**) and (**6.38**). Let $\mathtt{t}''$ denote the first round between $\mathtt{t}$ and $\mathtt{t}'$ such that one of these two lines is reached. Note that $\mathtt{t}'' > \mathtt{t}$, since (**6.33**) already passed by the start of Stage 2 (which is when the subclaim asserts $FR = \mathtt{t}_0$), and (**6.38**) cannot be reached in round $\mathtt{t}$ since OUT has problem status when (**6.37**) of round $\mathtt{t}$ is reached (by hypothesis).

- Suppose FR is *first* changed from $FR = \mathtt{t}_0$ on (**6.33**) of round $\mathtt{t}''$. First note that because (**6.33**) is the *first* time FR changes its value from $\mathtt{t}_0$, it must be the case that FR was still equal to $\mathtt{t}_0$ on (**6.30**) earlier in round $\mathtt{t}''$. Also, since (**6.33**) is reached in round $\mathtt{t}''$, OUT returns to normal status. Since $\mathtt{t}'$ was defined to be the first round after $\mathtt{t}$ for which this happens, we must have that $\mathtt{t}'' \geq \mathtt{t}'$. But by construction $\mathtt{t}'' \leq \mathtt{t}'$, so we must have that $\mathtt{t}'' = \mathtt{t}'$. However, this is a contradiction, because our assumption is that $FR \neq \mathtt{t}_0$ on line (**6.30**) of round $\mathtt{t}' = \mathtt{t}''$, but as noted in the second sentence of this paragraph, we are in the case that $FR = \mathtt{t}_0$ on line (**6.30**) of round $\mathtt{t}''$.
- Suppose FR is *first* changed from $FR = \mathtt{t}_0$ on (**6.38**) of round $\mathtt{t}''$. Then (**6.37**) must have been satisfied, and thus OUT had normal status when *Create Flagged Packet* was called in round $\mathtt{t}''$. Since OUT had problem status at the start of Stage 2 of round $\mathtt{t}$ (by hypothesis), the status must have switched to normal at some point between $\mathtt{t}$ and $\mathtt{t}''$, which can only happen on (**6.33**). But if (**6.33**) is reached, then FR will be set to $\perp$ on this line, which contradicts the fact that FR was first changed from $FR = \mathtt{t}_0$ on (**6.38**) of round $\mathtt{t}''$.

This completes the proof of the subclaim.

For the third Statement, first note that $\mathsf{OUT}[H_{FP}] = p$ as of line (**6.38**) of round $\mathtt{t}'$. This is the case since $sb_{OUT} = 0$ on line (**5.12**) (by Statement 1 of this claim), and then the fact that *Send Packet* is called in round $\mathtt{t}'$ means $H_{OUT} > H_{IN}$ on (**5.16**), and therefore since none of these values change between (**5.12**) and (**5.16**), (**6.37**) will be satisfied in round $\mathtt{t}'$. Therefore, we will track all changes to OUT and H_{FP} from Stage 2 of round $\mathtt{t}'$ through the time p is deleted from OUT as on (**6.32-33**) of some later round[19], and show that none of these changes will alter the fact that $\mathsf{OUT}[H_{FP}] = p$. Notice that (before the end of the transmission) H_{FP} only changes value on lines (**6.33**), (**6.35**), and (**6.38**); while OUT only changes values on lines (**6.32**), (**6.35**), and (**7.89-90**). Clearly the changes to each value on (**6.35**) will preserve $\mathsf{OUT}[H_{FP}] = p$, so it is enough to check the other changes. Notice that (**6.32**) is reached if and only if (**6.33**) is reached, which by Statement 2 of this claim does not happen until OUT gets confirmation of receipt that p was successfully received by B's neighbor, and therefore these changes also do not threaten the validity of Statement 3. The change to H_{FP} as on (**6.38**) can only occur if (**6.37**) is satisfied, i.e. only if OUT has normal status,

[19] Or through the end of the transmission, whichever occurs first.

and thus again Statement 2 of this claim says this cannot happen until OUT gets confirmation of receipt that p was successfully received by B's neighbor. Finally, lines (**7.89-90**) will preserve $\mathsf{OUT}[H_{FP}] = p$ by Statement 11 of Lemma 7.1.

That $FR = \mathtt{t}'$ from (**6.38**) of $\mathtt{t}'$ through the time B gets confirmation of receipt for p was proven in the subclaim above. Also, $\tilde{p}$ can only change on (**6.33**) or (**6.38**), which we already proved (in the proof of the subclaim above) are not reached. ∎

Corollary 7.8. *At any time, an outgoing buffer has at most one flagged packet.*

Proof. This follows immediately from Statement 3 of Lemma 7.7. ∎

Claim 7.9. *For any outgoing buffer* OUT*, if at any time its Flagged Round value* FR *is equal to* $\mathtt{t}$*, then* OUT *necessarily called Send Packet on line (***5.17***) of round* $\mathtt{t}$*.*

Proof. Suppose that at some point in time, FR is set to $\mathtt{t}$. Notice that the only place FR assumes non-$\perp$ values is on (**6.38**), and therefore line (**6.37**) must have been satisfied in round $\mathtt{t}$. Since the values for sb_{OUT}, H_{OUT}, and H_{IN} cannot change between lines (**5.15**) and (**5.16**), the statement on (**5.16**) will also be satisfied in round $\mathtt{t}$, and consequently *Send Packet* will be reached in $\mathtt{t}$. ∎

Lemma 7.10. *Suppose that* $sb_{OUT} = 1$ *when line (***6.47***) is reached in round* $\mathtt{t}$ *on an edge linking buffers* OUT *and* IN*. Further suppose that* IN *does receive the communication* (p, FR) *from* OUT *on line (***6.43***) of* $\mathtt{t}$*. Also, let* $\mathtt{t}_0$ *denote the round described by* FR*, let* h *denote the height of the packet in* OUT *in round* $\mathtt{t}_0$*, and let* h' *denote the height of* IN *at the start of round* $\mathtt{t}_0$*. Then the following are true:*

1. $\mathtt{t}_0$ *is well-defined (i.e.* $FR \neq \perp$ *and* $FR \leq \mathtt{t}$*).*
2. $h > h'$.
3. OUT *sent* p *to* IN *on line (***6.41***) of round* $\mathtt{t}_0$*. Furthermore, the height of* p *in* OUT *when it is sent on line (***6.41***) of round* $\mathtt{t}$ *is greater than or equal to* h*.*
4. *If the condition statement on line (***6.51***) of round* $\mathtt{t}$ *is satisfied, then the value of* H_{GP} *when this line is entered, which corresponds to the height in* IN *that* p *assumes when it is inserted, satisfies:* $\perp \neq H_{GP} \leq h' + 1 \leq 2n$*.*
5. *If the condition statement on line (***6.51***) of round* $\mathtt{t}$ *is satisfied, then* H_{IN} *was less than* $2n$ *at the start of all rounds between* $\mathtt{t}_0$ *and* $\mathtt{t}$*.*

Proof of Lemma 7.10. We make a series of Subclaims to prove the 5 statements of the Lemma.

Subclaim 1. The value of FR that is sent on (**6.41**) of round $\mathtt{t}$ is not $\perp$.

Proof. Since (**6.41**) is reached, *Send Packet* was called on (**5.17**). By Statement 2 of Claim 7.5, we have that $H_{FP} \geq 1$ when *Send Packet* is called, and in particular $H_{FP} \neq \perp$ on line (**5.17**). Since H_{FP} cannot change between (**5.17**) and (**6.41**), we have that $H_{FP} \neq \perp$ on (**6.41**), and hence $FR \neq \perp$ on this line (Claim 7.4).

Subclaim 2. $\mathtt{t}_0$ is well-defined (i.e. $\perp \neq \mathtt{t}_0 \leq \mathtt{t}$).

Proof. By the definition of $\mathtt{t}_0$ and Subclaim 1, $\mathtt{t}_0 \neq \perp$. Also, by looking at the three places that FR changes values ((**6.33**), (**6.38**), and (**6.62**)), it is clear that FR will always be less than or equal to the current round index.

Subclaim 3. $\mathtt{t} > \mathtt{t}_0$.

Proof. That $\mathtt{t} \geq \mathtt{t}_0$ is immediate (FR is reset to $\perp$ at the start of every transmission (**3.34**) and (**6.62**), after which time the FR can never attain a value *bigger* than the current round (**6.38**)). Therefore, we only have to show $\mathtt{t} \neq \mathtt{t}_0$. For the sake of contradiction, suppose $\mathtt{t} = \mathtt{t}_0$. By hypothesis, $sb_{OUT} = 1$ when line (**6.47**) of round $\mathtt{t} = \mathtt{t}_0$ is reached. Notice that sb_{OUT} is reset to 0 on (**5.10**) of round $\mathtt{t} = \mathtt{t}_0$, so the only way it can be '1' on (**6.47**) later that round is if it is set to one on (**5.11**). This can only happen if $H_{OUT} = \perp$ or $FR > RR$. Since (**6.47**) is reached, (**6.44**) must have failed, and since H_{OUT} does not change values between the time it is received on (**5.11**) and (**6.44**), we have that $H_{OUT} \neq \perp$ on (**5.11**). Therefore, we must have that $FR > RR$ on (**5.11**) of round $\mathtt{t} = \mathtt{t}_0$.

Notice the value for FR here comes from the value sent by OUT on (**5.05**), and this happens *before* line (**6.38**) has been reached in round $\mathtt{t} = \mathtt{t}_0$. Therefore, the value of FR received on (**5.11**) obeys $FR < \mathtt{t} = \mathtt{t}_0$ (as noted above, FR can never attain a value *bigger* than the current round). Since $RR < FR$, line (**6.30**) *cannot* have been satisfied since the time FR was set to its current value (within a transmission, the values RR assumes are strictly *increasing*, see (**3.34**), (**6.53**), and (**6.64**)). Therefore, we may apply Claim 7.9 and Claim 7.7 to argue that FR will not be changed on (**6.38**) of round $\mathtt{t} = \mathtt{t}_0$ (since OUT will have problem status), and consequently FR will still be strictly smaller than $\mathtt{t} = \mathtt{t}_0$ when line (**6.41**) is reached of round $\mathtt{t}_0$. This contradicts the definition of $\mathtt{t}_0$ as the value received on line (**6.43**) of round $\mathtt{t}$.

Subclaim 4. OUT had normal status at the start of Stage 2 of round $\mathtt{t}_0$. For every round between Stage 2 of $\mathtt{t}_0 + 1$ through $\mathtt{t} - 1$, OUT had problem status and $FR = \mathtt{t}_0$.

Proof. By definition of $\mathtt{t}_0$, it equals the value of FR that was received in round $\mathtt{t}$ on line (**6.43**), which in turn corresponds to the value of FR that was sent on line (**6.41**). Tracing the values of FR backwards, we see that the only time/place FR is set to a non-$\perp$ value (as we know it has by Subclaim 1) is on line (**6.38**), and this must have happened in round $\mathtt{t}_0$ since $FR = \mathtt{t}_0$ by definition of $\mathtt{t}_0$. Therefore, in round $\mathtt{t}_0$, line (**6.38**) must have been reached when *Create Flagged Packet* was called on line (**5.15**); so in particular sb_{OUT} must have been zero on line (**6.37**) to have entered the conditional statement. Since sb_{OUT} cannot change between the start of Stage 2 and line (**5.15**) (where *Create Flagged Packet* is called), it must have been zero at the start of Stage 2. This proves the first part of the subclaim. Now suppose there is a round $\mathtt{t}'$ between Stage 2 of $\mathtt{t}_0 + 1$ and $\mathtt{t} - 1$ such that $sb_{OUT} = 0$ at any time in that round (without loss of generality, let $\mathtt{t}'$ be the first such round). Since sb_{OUT} can only switch to zero on (**6.33**) inside the call to *Reset Outgoing Variables*, it must be that this line is reached in $\mathtt{t}'$, and hence FR is also set to $\perp$ on this line. Since FR is only assigned non-$\perp$ values on (**6.38**), FR can only assume values at least $\mathtt{t}' > \mathtt{t}_0$ after this point. Thus, FR will not ever be able to return to the value of $\mathtt{t}_0$, contradicting the fact that $FR = \mathtt{t}_0$ during round $\mathtt{t}$. By the same reasoning, FR can never change value from $\mathtt{t}_0$ between the rounds $\mathtt{t}_0$ and $\mathtt{t}$.

Subclaim 5. OUT attempted to send p in round $\mathtt{t}_0$.

Proof. By definition, $\mathtt{t}_0$ denotes the value of FR during round $\mathtt{t}$. Since FR can only be set to $\mathtt{t}_0$ on (**6.38**) of round $\mathtt{t}_0$, this line must have been reached in $\mathtt{t}_0$. In particular, line (**6.37**) was satisfied during the call to *Create Flagged Packet* of round $\mathtt{t}_0$, and hence $sb = 0$ and $H > H_{IN}$ at that time. Therefore, (**5.16**) will be satisfied when it is reached in round $\mathtt{t}_0$, which implies *Send Packet* will be called on the following line. The fact that it was the same packet p that was sent in $\mathtt{t}_0$ as in $\mathtt{t}$ follows from Statement 3 of Lemma 7.7.

Subclaim 6. The height of p in OUT when it is transferred in round $\mathtt{t}$ is greater than or equal to h.

Proof. Subclaim 5 stated that OUT attempted to send p in round $\mathtt{t}_0$, and Subclaim 4 stated that OUT had normal status at the start of $\mathtt{t}_0$. Therefore, the packet which was sent in round $\mathtt{t}_0$ (which is p) was initialized inside the call to *Create Flagged Packet* on line (**6.38**). By observing the code there, we see that p is set to $\mathsf{OUT}[H]$, i.e. p has height H in round $\mathtt{t}_0$, and H_{FP} is set to equal H on this same line. By Statement 3 of Claim 7.7, $p = \tilde{p}$ will remain the flagged packet through the start of round $\mathtt{t}$, and $\mathsf{OUT}[H_{FP}] = p$. By Statement 11 of Lemma 7.1, H_{FP} will not change during any call to re-shuffle. Indeed, since Subclaim 4 ensures that line (**6.38**) is never reached from $\mathtt{t}_0 + 1$ through the start of $\mathtt{t}$, the only place H_{FP} can change value is on (**6.33**) or (**6.35**). We know the former cannot happen between $\mathtt{t}_0 + 1$ and the start of $\mathtt{t}$, since this would imply sb_{OUT} is re-set to zero on (**6.33**) of that round, contradicting Subclaim 4. Therefore, H_{FP} can only change values between $\mathtt{t}_0 + 1$ and the start of $\mathtt{t}$ as on (**6.35**), which can only *increase* H_{FP}. Hence, from the time H_{FP} is set to equal the height of OUT in round $\mathtt{t}_0$ as on (**6.38**) (which by definition is h), H_{FP} can only increase through the start of round $\mathtt{t}$.

Subclaim 7. $h > h'$.

Proof. This follows immediately from Subclaims 4 and 5 as follows. Because OUT tried to send the packet in round $\mathtt{t}_0$ (Subclaim 5) and because OUT had normal status in this round (Subclaim 4), it must be that the conditional statement on line (**5.16**) of round $\mathtt{t}_0$ was satisfied, and in particular that the expression $H > H_{IN}$ was true. Since h is defined to be the value of H, H_{FP} as of line (**6.38**) of round $\mathtt{t}_0$ (Statement 6 of Lemma 7.1), this subclaim will follow if h' equals the value of H_{IN} as of line (**6.38**) of round $\mathtt{t}_0$. But this is true by Statement 5 of Lemma 7.1, since the value of H_{IN} on line (**5.16**) comes from the value received on line (**5.06**), which in turn corresponds to the value of H_{IN} sent on line (**5.09**).

Subclaim 8. If the conditional statement on line (**6.51**) is satisfied in round $\mathtt{t}$, then OUT's attempt to send p in round $\mathtt{t}_0$ failed (i.e. IN did not store p in $\mathtt{t}_0$), and furthermore IN did not store p in any round between $\mathtt{t}_0$ and $\mathtt{t}$.

Proof. We prove this by contradiction. Suppose there is some round $\widetilde{\mathtt{t}} \in [\mathtt{t}_0..\mathtt{t} - 1]$ in which IN stored p. This would mean that line (**6.51**) was satisfied in round $\widetilde{\mathtt{t}}$, and in particular RR is set to $\widetilde{\mathtt{t}} \geq \mathtt{t}_0$ on (**6.53**). But as already noted in the proof of Subclaim 2, for the remainder of the transmission, FR can never assume the value of a round *before* $\mathtt{t}_0$. Similarly, once RR changes to $\widetilde{\mathtt{t}} \geq \mathtt{t}_0 \geq FR$ on (**6.53**) of round $\widetilde{\mathtt{t}}$, it can never assume a smaller (non-$\perp$) value for the rest of the transmission (RR can only change to a non-$\perp$ value on line (**6.53**)). But this contradicts the fact that $RR < FR$ on (**6.51**) of round $\mathtt{t}$.

Subclaim 9. If the conditional statement on line (**6.51**) is satisfied in round $\mathtt{t}$, then $RR < \mathtt{t}_0$

between the start of $\mathtt{t}_0$ through line (**6.51**) of round $\mathtt{t}$. In particular, lines (**6.47**) and (**6.51**) will be satisfied for any round between $\mathtt{t}_0$ and $\mathtt{t}$ for which they are reached.

Proof. RR is set to -1 at the start of any transmission ((**3.31**) and (**6.64**)). Since the only other place RR changes value is (**6.53**), it is always the case that the value of RR is less than or equal to the index of the current round. Thus, RR can only assume a value greater than (or equal to) $\mathtt{t}_0$ in a round *after* (or *during*) $\mathtt{t}_0$. But this would mean there was some round between $\mathtt{t}_0$ and $\mathtt{t}-1$ (inclusive) such that (**6.53**) was reached, which contradicts Subclaim 8. The fact that (**6.51**) will be satisfied whenever it is reached now follows immediately from Statement 3 of Claim 7.7, since in order to reach (**6.51**), line (**6.48**) must have failed, which means the communication on line (**6.43**) was received. The fact that (**6.47**) will be satisfied whenever it is reached follows from the fact that sb_{OUT} will always be set to one on (**5.11**) of each round between $\mathtt{t}_0$ and $\mathtt{t}$ (the first part of this subclaim says $RR < \mathtt{t}_0$, and Subclaim 4 says that if FR is received on (**5.11**), then $FR = \mathtt{t}_0$).

Subclaim 10. If the conditional statement on line (**6.51**) is satisfied in round $\mathtt{t}$, then there was no round between $\mathtt{t}_0+1$ and $\mathtt{t}-1$ (inclusive) in which IN received both H_{OUT} and p.

Proof. Suppose for the sake of contradiction that there is such a round, $\widetilde{\mathtt{t}}$. Notice that line (**6.51**) of round $\widetilde{\mathtt{t}}$ will necessarily be reached (since the conditional statement of line (**6.44**) will fail by assumption, (**6.47**) will be satisfied by Subclaim 4, and (**6.48**) will fail by assumption). However, line (**6.53**) cannot be reached in round $\widetilde{\mathtt{t}}$ (Subclaim 8 above), and therefore the conditional statement on line (**6.51**) must fail. This contradicts Subclaim 9.

Subclaim 11. If the conditional statement on line (**6.51**) is satisfied in round $\mathtt{t}$, then IN was in problem status at the end of round $\mathtt{t}_0$, and remained in problem status until line (**6.53**) of round $\mathtt{t}$.

Proof. We first show that sb_{IN} will be set to one on line (**6.45**) or (**6.49**) of round $\mathtt{t}_0$. To see this, we note that if (**6.44**) fails in round $\mathtt{t}_0$, then necessarily (**6.47**) and (**6.48**) will both be satisfied. Afterall, (**6.47**) is satisfied (Subclaim 7), and if (**6.48**) *failed*, then (**6.51**) would be reached and subsequently satisfied (Subclaim 9), which would contradict Subclaim 8. For every round between $\mathtt{t}_0+1$ and $\mathtt{t}$, we will show that either the conditional statement on line (**6.44**) will be satisfied, or the conditional statements on lines (**6.47**) and (**6.48**) will both be satisfied, and hence sb_{IN} can never be reset to zero since lines (**6.53**), (**6.55**), and (**6.57**) will never be reached. To see this, let $\mathtt{t}' \in [\mathtt{t}_0+1..\mathtt{t}-1]$. If (**6.44**) is satisfied for $\mathtt{t}'$, then we are done. So assume (**6.44**) is *not* satisfied for $\mathtt{t}'$, and hence IN did *not* receive the communication on (**6.43**) (Subclaim 10). This means (**6.48**) will be satisfied. The fact that (**6.47**) is also satisfied follows from Subclaim 9.

Subclaim 12. If the conditional statement on line (**6.51**) is satisfied in round $\mathtt{t}$, then between the end of round $\mathtt{t}_0$ and the time *Receive Packet* is called in round $\mathtt{t}$, we have that $H_{GP} \neq \perp$ and $H_{GP} \leq h'+1 \leq 2n$.

Proof. As in the proof of Subclaim 11, either line (**6.46**) or (**6.50**) will be reached in round $\mathtt{t}_0$ (since either line (**6.45**) or (**6.49**) is reached). The value of H_{IN} at the start of round $\mathtt{t}_0$ is h' by definition. Since $h' < h \leq 2n$ (the first inequality is Subclaim 7, the second is Statements 6 and 9 of Lemma 7.1), and since H_{IN} cannot change value between the start of

$\mathtt{t}_0$ and the time *Receive Packet* is called, we have that the value of $H_{IN} < 2n$ when either line (**6.46**) or (**6.50**) is reached. Therefore, these lines guarantee that $\perp \neq H_{GP} \leq h' + 1 \leq 2n$ after these lines. After this, there are five places H_{GP} can change its value: (**6.46**), (**6.50**), (**6.53**), (**6.55**), and (**6.57**). As in the proof of Subclaim 11, lines (**6.55**) and (**6.57**) will not be reached at any point between $\mathtt{t}_0$ and $\mathtt{t}$, nor will line (**6.53**) by Subclaim 8. The other two lines that change H_{GP} can only *decrease* it (but they cannot set H_{GP} to $\perp$).

Subclaim 13. If the condition statement on line (**6.51**) of round $\mathtt{t}$ is satisfied, then the value of H_{GP} when this line is entered, which corresponds to the height in IN that p assumes when it is inserted, satisfies: $H_{GP} \neq \perp$ and $H_{GP} \leq h' + 1 \leq 2n$.

Proof. This follows immediately from Subclaim 12 since p is inserted into IN at height H_{GP} (**6.53**).

Subclaim 14. If the condition statement on line (**6.51**) of round $\mathtt{t}$ is satisfied, then H_{IN} was less than $2n$ at the start of all rounds between $\mathtt{t}_0$ and $\mathtt{t}$.

Proof. Subclaim 12 implies that $h' < 2n$ (so H_{IN} had height strictly smaller than $2n$ at the start of round $\mathtt{t}_0$). Searching through the pseudo-code, we see that H_{IN} is only modified on lines (**6.53**), and during Re-Shuffling (**7.91-92**). Between rounds $\mathtt{t}_0$ and $\mathtt{t}$, line (**6.53**) is never reached (Subclaim 8), and hence all changes to H_{IN} must come from Re-Shuffling. But because H_{IN} was less than $2n$ when it entered the Re-Shuffle phase in round $\mathtt{t}_0$, Statement 13 of Lemma 7.1 guarantees that H_{IN} will still be less than $2n$ at the start of round $\mathtt{t}$.

All Statements of the Lemma have now been proven. ■

Claim 7.11. *Every packet is inserted into one of the sender's outgoing buffers at some initial height. When (a copy of) the packet goes between any two buffers $B_1 \neq B_2$ (either across an edge or locally during re-shuffling), its height in B_2 is less than or equal to the height it had in B_1. If $B_1 = B_2$, the statement remains true EXCEPT for on line (**6.35**).*

Proof. We separate the proof into cases, based on the nature of the packet movement. The only times packets are accepted by a new buffer or re-shuffled within the same buffer occurs on lines (**6.32**), (**6.35**), (**6.53**), (**6.55**), (**6.57**), (**6.61**), (**6.64**), (**7.89-90**), and (**7.101-102**). Of these, (**6.35**) is excluded from the claim, and the packet movement on lines (**6.32**), (**6.55**), (**6.57**), (**6.61**), (**6.64**), and (**7.101-102**) are all clearly strictly downwards. It remains to consider lines (**6.53**) and (**7.89-90**).

CASE 1: THE PACKET MOVED DURING RE-SHUFFLING AS ON (**7.89-90**). By investigating the code on these lines, we must show that $m+1 \leq M$. This was certainly true as of line (**7.74**), but we need to make sure this didn't change when *Adjust Heights* was called. The changes made to M and m on (**7.83**) and (**7.85**) will only serve to help the inequality $m+1 \leq M$, so we need only argue the cases for when (**7.81**) and/or (**7.87**) is reached. Notice that if either line is reached, by (**7.74**) we must have (before adjusting M and m) that $M - m \geq 2$, and therefore modifying *only* $M = M - 1$ *or* $m = m + 1$ won't threaten the inequality $m + 1 \leq M$. It remains to argue that both (**7.81**) **and** (**7.87**) cannot happen simultaneously (i.e. cannot both happen within the same call to *Re-Shuffle*). If both of these were to happen, then it must be that during this call to *Re-Shuffle*, there was an outgoing buffer B_1 that had height 2 or more higher than an incoming buffer B_2 (see lines (**7.72-74**) and (**7.80**) and (**7.86**)). We argue that this cannot ever happen. By Claim 6.3, at the end of the previous round, we had that the height of B_1 was at most one bigger than the height of B_2. During routing, B_2 can only get bigger and B_1 can only get smaller ((**6.53**) and (**6.33**) are the only places these heights change). Therefore, after Routing but before any Re-Shuffling, we have again that the height of B_1 was at most one bigger than the height of B_2. Therefore, in order for B_1 to get at least 2 bigger than B_2, either a packet must be shuffled *into* B_1, or a packet must be shuffled *out of* B_2, and this must happen when B_1 is already one bigger than B_2. But analyzing (**7.72**) and (**7.73**) shows that this can never happen.

CASE 2: THE PACKET MOVED DURING ROUTING AS ON (**6.53**). In order to reach (**6.53**), the conditional statements on lines (**6.47**), (**6.48**), and (**6.51**) all must be satisfied, so $p \neq \perp$, $RR < FR$, and either $sb_{OUT} = 1$ or $H_{OUT} > H$ (or both). We investigate each case separately:

CASE A: $sb_{OUT} = 1$ ON LINE (**6.47**). Then Statements 2-4 of Lemma 7.10 imply that the height of the packet in B_1 is greater than or equal to the height it will be stored into in B_2, as desired.

CASE B: $sb_{OUT} = 0$ AND $H_{OUT} > H_{IN}$ ON LINE (**6.47**). For notational convenience, denote the current round (when the hypotheses of Case B hold) by $\mathtt{t}$. First note that Statements 1 and 2 of Lemma 7.1 imply that the height the packet assumes in B_2 (H_{GP}) is less than or equal to $H_{IN}+1$. Meanwhile, since $sb_{OUT} = 0$ (it is set on (**5.11**) of round $\mathtt{t}$), the value received for H_{OUT} on (**5.11**) is not $\perp$, and the value for FR received on (**5.11**) is either $\perp$ or satisfies $FR \leq RR$. Notice that the case $FR \leq RR$ is not possible, since then (**6.53**) would not be reached ((**6.51**) would fail). Therefore, $FR = \perp$ but $H_{OUT} \neq \perp$, and so B_2 received the communication sent by B_1 on (**5.05**) of round $\mathtt{t}$, which had the first of the two possible forms. In particular, $H_{FP} = \perp$ at the outset of $\mathtt{t}$, and since H_{FP} cannot change between the start of a round a line (**6.38**) of the previous round, we must have that (**6.37**) failed in round $\mathtt{t}-1$. By this fact and Claim 7.4, B_1 had normal status when (**5.16**) was reached in round $\mathtt{t}-1$, and this will not be able to change in the call to *Reset Outgoing Variables* of round $\mathtt{t}$ because $d = 0$ (**6.25**) (since d is reset to zero every round on (**6.26**), it can only have non-zero values between line (**6.40**) of one round and line (**6.26**) of the following round IF a packet was sent the earlier round. However, as already noted this did not happen, as the fact that OUT had normal status and yet (**6.37**) failed in round $\mathtt{t}-1$ implies that (**5.16**) will also fail in round $\mathtt{t}-1$). Therefore, B_1 has normal status when *Create Flagged Packet* is called in round $\mathtt{t}$, and in particular, H_{FP} is set to H_{OUT} on (**6.38**), i.e. the flagged packet to be transferred during $\mathtt{t}$ has height H_{OUT} in B_1. Putting this all together, the packet has height H_{OUT} in B_1 and assumes height H_{GP} in B_2. But as argued above, $H_{OUT} \geq H_{IN} + 1 \geq H_{GP}$, as desired. ∎

Claim 7.12. *Before* ***End of Transmission Adjustments*** *is called in any transmission* $\mathtt{T}$ *(**6.61**), any packet that was inserted into the network during transmission* $\mathtt{T}$ *is either in some buffer (perhaps as a flagged packet) or has been received by R.*

Proof. As packets travel between nodes, the sending node maintains a copy of the packet until it has obtained verification from the receiving node that the packet was accepted. This way, packets that are lost due to edge failure are backed-up. This is the high-level idea of why the claim is true, we now go through rigorous detail.

First notice that the statement only concerns packets corresponding to the current codeword transmission, and packets deleted as on (**6.61**) do not threaten the validity of the Claim. We consider a specific packet p that has been inserted into the network and show that p is never removed from a buffer B until another buffer B' has taken p from B. We do this by considering every line of code that a buffer could possible remove p, and argue that whenever this happens, p has necessarily been accepted from B by some other buffer B'. Notice that the only lines that a buffer could possibly remove p (before line (**6.61**) of T is reached) are: (**6.32**), (**6.53**), and (**6.89-90**).

Line (**6.53**). This line is handled by Lemma 7.1, Statements 1 and 2, which say that whenever a slot of an incoming buffer is filled as on line (**6.53**), it fills an empty slot, and therefore cannot correspond to removing (over-writing) p.

Lines (**7.89-90**). These lines are handled by Lemma 7.1, Statement 10.

Line (**6.32**). This is the interesting case, where p is removed from an outgoing buffer after a packet transfer. We must show that any time p is removed here, it has been accepted by some incoming buffer B'. For notation, we will let $\mathtt{t}$ denote the round that p is deleted from B (i.e. when line (**6.32**) is reached), and $\mathtt{t}_0$ denote the round that B first tried to send the packet to B' as on (**6.41**). By Statement 3 of Claim 7.7, $\mathtt{t}_0$ is the round that $\tilde{p}$ was most recently set to p as on line (**6.38**) (note that $\mathtt{t}_0 \leq \mathtt{t}$). Since line (**6.32**) was reached in round $\mathtt{t}$, the conditional statements on lines (**6.29**) and (**6.30**) were satisfied, and so $\perp \neq RR \geq FR$ when those lines were reached. By Statement 3 of Claim 7.7, FR will equal $\mathtt{t}_0$ when (**6.30**) is satisfied. Since in *any* round $\mathtt{t}'$, the only non-$\perp$ value that RR can ever be set to is $\mathtt{t}'$ (**6.53**), and since $RR \geq \mathtt{t}_0 = FR$ (**6.30**), it must be that (**6.53**) was reached in some round $\mathtt{t}' \in [\mathtt{t}_0, \mathtt{t}]$. In particular, B' stored a packet as on (**6.53**) of round $\mathtt{t}'$, which by Statement 3 of Claim 7.7 was necessarily p. ■

Claim 7.13. *Not counting flagged packets, there is at most one copy of any packet in the network at any time (not including packets in the sender or receiver's buffers). Looking at all copies (flagged and un-flagged) of any given packet present in the network at any time, at most one copy of that packet will* **ever** *be accepted (as in Definition 6.5) by another node.*

Proof. For any packet p, let N_p denote the copies of p (both flagged and not) present in the network (in an internal node's buffer) at a given time. We begin the proof via a sequence of observations:

Observation 1. *The only time N_p can ever* ***increase*** *is on line (***6.53***).*

Proof. The only way for N_p to increase is if (a copy of) p is stored by a new buffer. Looking at the pseudo-code, the only place a buffer slot can be assigned a new copy of p is on lines (**6.32**), (**6.35**), (**6.53**), (**6.55**), (**6.57**), (**6.61**), (**6.64**), and (**7.89**). Of these, only (**6.53**) and (**7.89**) could possibly increase N_p, as the others simply shift packets within a buffer and/or delete packets. In the latter case, N_p does not change by Statement 10 of Lemma 7.1.

Observation 2. *Suppose A (including $A = S$) first sends a (copy of a) packet p to B as on (***6.41***) of round $\mathtt{t}_0$. Then:*

*(a) The copy of p in A's outgoing buffer along $E(A, B)$ (for which there was a copy made and sent on (***6.41***) of round $\mathtt{t}_0$) will never be transferred to any of A's other buffers.*

(b) *The copy of p will remain in A's outgoing buffer along $E(A, B)$ as a flagged packet until it is deleted either when A gets confirmation of receipt (see Definition 7.6) in some round* $\texttt{t}$ *(**6.32**), or by the end of the transmission as on (**6.61**). In the latter case, define* $\texttt{t} := 3D$ *(the last round of the transmission) for Statement (c) below.*

(c) *Between* $\texttt{t}_0$ *and line (**5.07**) of round* $\texttt{t}$*, B will accept (a copy of) p from A as on (**6.53**) at most once. Furthermore, the copy of p in A's buffer cannot move to any other buffer or generate any other copies other than the one (possibly) received by B as on (**6.53**).*

Proof. Statement (a) follows from Statement 3 of Claim 7.7 and Statement 11 of Lemma 7.1, together with the fact that lines (**6.32**) and (**6.61**, **6.69**) imply that the relevant copy of A will be deleted when it does get confirmation of receipt as in Definition 7.6 (or the end of the transmission). By Statement 3 of Claim 7.7, this copy of p will be (the unique) flagged packet in A's outgoing buffer to B until confirmation of receipt (or the end of the transmission), which proves Statement (b). For Statement (c), suppose that B accepts a copy of p as on (**6.53**) during some round $\texttt{t}' \in [\texttt{t}_0, \texttt{t}]$. Then RR will be set to $\texttt{t}'$ on (**6.53**) of round $\texttt{t}'$, and RR cannot obtain a *smaller* index until the next transmission (**6.53**). By Statement 3 of Claim 7.7, FR will remain equal to $\texttt{t}_0$ from line (**6.38**) of round $\texttt{t}_0$ through the time (**6.33**) of round $\texttt{t}$ is reached. Therefore, between $\texttt{t}' \geq \texttt{t}_0$ and line (**6.33**) of round $\texttt{t}$, we have that $FR = \texttt{t}_0 \leq \texttt{t}' \leq RR$, and hence line (**6.51**) can never be satisfied during these times, which implies (**6.53**) can never be reached again after $\texttt{t}'$. This proves the first part of Statement (c). The second part follows by looking at all possible places (copies of) packets can move or be created: (**6.32**), (**6.35**), (**6.53**), (**6.55**), (**6.57**), (**6.61**), (**6.64**), and (**7.89-90**). Of these, only (**6.53**) and (**7.89-90**) threaten to move p or create a new copy of p. However, the first part of Observation 2(c) says that (**6.53**) can happen at most once (and is accounted for), while Statement 11 of Lemma 7.1 rules out the case that the packet is re-shuffled as on (**7.89-90**).

Observation 3. *No packet will ever be inserted (see Definition 6.6) into the network more than once. In particular, for any packet p, $N_p = 0$ until the sender inserts it (i.e. some node accepts the packet from the sender as on (**6.53**)), at which point $N_p = 1$. After this point, the only way N_p can become larger than one is if (**6.53**) is reached, where neither the sending node nor the receiving node is S or R.*

Proof. Since the packets of S are distributed to his outgoing buffers before being inserted into the network (**4.38**), (**6.65**), and (**6.67-70**), and since S never receives a packet he has already inserted (S has no incoming buffers (**3.17**)) nor shuffles packets between buffers ((**5.22**) and (**7.95-96**)), a given packet p can only be insereted along one edge adjacent to the sender. The fact the sender can insert at most one (copy of a) packet p along an adjacent edge now follows from Observation 2 above for $A = S$. This proves the first part of Observation 3.

By Observation 1, the only place N_p can increase is on (**6.53**). Whenever this line is reached, the copy stored comes from the one received on (**6.43**), which in turn was sent by another node on (**6.41**). The copy sent on (**6.41**) in turn can only be set on (**6.38**) (perhaps in an earlier round), so in particular a copy of the packet must have already existed in an outgoing buffer of the sending node. This proves that when N_p goes from zero to one, it can only happen when a packet is inserted for the first time by the sender. The rest of Observation 3 now follows from Observation 1, the first part of Observation 3, the fact that copies reaching

R do not increase N_p (by definition of N_p), and the fact R never sends a copy of a packet (**3.07**) and S never accepts packets (**3.17**).

Define a copy of a packet p in the network to be *dead* if that copy will *never* leave the buffer it is currently in, nor will it ever generate any new copies. A copy of a packet that is not dead will be *alive*.

Observation 4. *If a (copy of a) packet is ever flagged and dead, it will forever remain both flagged and dead, until it is deleted.*

Proof. By definition of being "dead," once a (copy of a) packet becomes dead it can never become alive again. Also, copies of a packet that are flagged remain flagged until they are deleted by Observation 2(b).

The Claim now follows immediately from the following subclaim:

Subclaim. *Fix any packet p that is ever inserted into the network. Then at any time, there is at most one* ***alive*** *copy of p in the network at any time. Also at any time, if there is one alive copy of p, then all* ***dead*** *copies of p are flagged packets. If there are no alive copies, then there is at most one dead copy of p that is not a flagged packet.*

Proof. Before p is inserted into the network, $N_p = 0$, and there is nothing to show. Suppose p is inserted into the network in round $\mathtt{t}_0$, so that $N_p = 1$ by the end of the round (Observation 3). Since $N_p = 1$, the validity of the subclaim is not threatened. Also, if this packet is *dead*, then the proof is complete, as by Observation 3 and the definition of *deadness*, no other (copies) of p will ever be created, and hence the subclaim will forever be true for p. So suppose p is *alive* when it is inserted. We will show that a (copy of an) alive packet can create at most one new (copy of a) packet, and the instant it does so, the original copy is necessarily both flagged and dead (the new copy may be either alive or dead), from which the subclaim follows from Observation 4. So suppose an alive copy of p creates a new copy (increasing N_p) of itself in round $\mathtt{t}$. Notice that the only time new copies of any packet can be created is on (**6.53**) (see e.g. proof of Observation 2). Fix notation, so that the alive copy of p was in node A's outgoing buffer to node B, and hence it was B's corresponding buffer that entered (**6.53**) in round $\mathtt{t}$. The fact that the alive copy of p in A's outgoing buffer is flagged and dead the instant B accepts it on (**6.53**) of round $\mathtt{t}$ follows immediately from Observation 2. ■

Lemma 7.14. *Suppose that in round* $\mathtt{t}$*, B accepts (as in Definition 6.5) a packet from A. Let* $O_{A,B}$ *denote A's outgoing buffer along* $E(A,B)$*, and let O denote the height the packet had in* $O_{A,B}$ *when* ***Send Packet*** *was called in round* $\mathtt{t}$ *(***5.17***). Also let* $I_{B,A}$ *denote B's incoming buffer along* $E(A,B)$*, and let I denote the height of* $I_{B,A}$ *at the start of* $\mathtt{t}$*. Then the change in non-duplicated potential caused by this packet transfer is less than or equal to:*

$$-O + I + 1 \qquad OR \qquad -O \ \ (\textit{if } B = R) \tag{5}$$

Furthermore, after the packet transfer but before re-shuffling, $I_{B,A}$ *will have height* $I + 1$*.*

Proof. By definition, B accepts the packet in round $\mathtt{t}$ means that (**6.53**) was reached by B's incoming buffer along $E(A,B)$ in round $\mathtt{t}$. Since the packet is stored at height H_{GP} (**6.53**), B's non-duplicated potential will increase by H_{GP} due to this packet transfer (if $B = R$, then by definition of non-duplicated potential, packets in R do not contribute anything, so there will be no change). By Statements 1 and 2 of Lemma 7.1, $H_{GP} \leq I+1$, and hence B's increase in non-duplicated potential caused by the packet transfer is at most $I+1$ (or zero in the case $B=R$). Also, since B had height I at the start of the round, and B accepts a packet on (**6.53**) of round $\mathtt{t}$, B will have $I+1$ packets in I when the re-shuffling phase of round $\mathtt{t}$ begins, which is the second statement of the lemma.

Meanwhile, the packet transferred along $E(A,B)$ in round $\mathtt{t}$ still has a copy in $O_{A,B}$ (until A receives confirmation of receipt from B, see Definition 7.6), but by definition of non-duplicated potential (see the paragraph between Claim 6.9 and Lemma 6.11), this (flagged) packet will no longer count towards non-duplicated potential the instant B accepts it as on (**6.53**) of round $\mathtt{t}$. Therefore, A's non-duplicated potential will drop by the value H_{FP} has when B accepts the packet on (**6.53**) (Statement 3 of Claim 7.7), which equals O since H_{FP} cannot change between the time *Send Packet* is called on (**5.17**) and the time the packet is accepted on (**6.53**). Therefore, counting only changes in non-duplicated potential due to the packet transfer, the change in potential is: $-O+H_{GP} \leq -O+I+1$ (or $-O$ in the case $B=R$), as desired. ∎

We now re-state and prove Lemma 6.14.

Lemma 7.15. *Let $\mathcal{C}=N_1N_2\dots N_l$ be a path consisting of l nodes, such that $R=N_l$ and $S\notin\mathcal{C}$. Suppose that in round $\mathtt{t}$, all edges $E(N_i,N_{i+1})$, $1\leq i<l$ are active for the entire round. Let ϕ denote the change in the network's non-duplicated potential caused by:*

1. *(For $1\leq i<l$) Packet transfers across $E(N_i,N_{i+1})$ in round $\mathtt{t}$,*
2. *(For $1<i<l$) Re-shuffling packets **into** N_i's outgoing buffers during $\mathtt{t}$,*

Then if O_{N_1,N_2} denotes N_1's outgoing buffer along $E(N_1,N_2)$ and O denotes its height at the start of $\mathtt{t}$, we have:

- *If O_{N_1,N_2} has a flagged packet that has already been accepted by N_2 **before** round $\mathtt{t}$, then:*

$$\phi \leq -O+l-1 \tag{6}$$

- *Otherwise,*

$$\phi \leq -O+l-2 \tag{7}$$

Proof. (Induction on l).
BASE CASE: $l=2$. So $\mathcal{C}=N_1R$.

Case 1: $O_{N_1,R}$ had a flagged packet at the start of $\mathtt{t}$ that was already accepted by N_2. Our aim for this case is to prove (6) for $l=2$. If $O<2$, then $-O+l-1\geq -1+2-1=0$, and then (6) will be true by Statement 3 of Lemma 6.11. So assume $O\geq 2$. Since $E(N_1,R)$ is active during $\mathtt{t}$ and R had already accepted the packet in some previous round $\tilde{\mathtt{t}}<\mathtt{t}$, we have that $RR\geq\tilde{\mathtt{t}}$ (**6.53**), and N_1 will receive this value for RR in R's stage one communication (**5.06**), (**5.09**). By Statment 3 of Claim 7.7, $FR\leq\tilde{\mathtt{t}}\leq RR$, and thus lines (**6.29-30**) will be satisfied in round $\mathtt{t}$, deleting the flagged packet on (**6.32**) and setting $sb=0$. When *Create Flagged Packet* is called on (**5.15**), a new packet will be flagged, with $H_{FP}=H_{OUT}=O-1$ and $FR=\mathtt{t}$ (since $O\geq 2$, there will be at least one packet left in $O_{N_1,R}$ of height $O-1>0$ by Lemma 7.1). Letting I denote the height of the receiver's incoming buffer along $E(N_1,R)$, we have that $I = 0$ (Claim 7.3). Therefore, $H_{OUT} > H_{IN}$, and so the flagged packet will be sent as on (**5.17**). Since R will receive and store this packet (since the edge is active and $RR < \mathtt{t} = FR$, lines (**6.44**) and (**6.48**) will fail, while lines (**6.47**) and (**6.51**) will be satisfied), we apply Lemma 7.14 to argue there will be a change in non-duplication potential that is less than or equal to $-(O-1)$, which is (6) (for $l = 2$).

Case 2: Either $O_{N_1,R}$ has no flagged packet at the start of $\mathtt{t}$, or if so, it has not yet been accepted by R. Our aim for this case is to prove (6) for $l = 2$. If $O = 0$, then $-O + l - 2 = 0$, and (7) is true by Statement 3 of Lemma 6.11. So assume $O \geq 1$. Then necessarily a packet will be sent during round $\mathtt{t}$ ((**5.16**) is necessarily satisfied since by assumption $E(N_1, R)$ is active during $\mathtt{t}$, $H_{OUT} \geq 1$ by Lemma 7.1 and $H_{IN} = 0$ by Claim 7.3). We first show that the height of the packet in $O_{N_1,R}$ that will be transferred in round $\mathtt{t}$ (which will be the value held by H_{FP} when *Send Packet* is called in round $\mathtt{t}$) is greater than or equal to O (whether or not it was flagged before round $\mathtt{t}$):

- If $O_{N_1,R}$ did not have any flagged packets at the outset of $\mathtt{t}$, then $H_{FP} = \perp$ at the start of $\mathtt{t}$, and so $sb = 0$ and $FR = \perp$ at the start of $\mathtt{t}$ by Claim 7.4. Since H_{FP} cannot change between the call to *Send Packet* in the previous round and the call to *Reset Outgoing Variables* in the current round, Statement 2 of Claim 7.5 implies no packet was sent the previous round, and hence $d = 0$ at the start of $\mathtt{t}$ (d was necessarily zero as of (**6.26**) of round $\mathtt{t}-1$, and as argued did not change to '1' on (**6.40**) later that round). Consequently, sb will remain zero from the start of $\mathtt{t}$ through the time *Create Flagged Packet* is called in round $\mathtt{t}$, and because $H_{OUT} = O > 0 = I = H_{IN}$, (**6.38**) will be reached in round $\mathtt{t}$, setting H_{FP} to O.
- Alternatively, if $O_{N_1,R}$ *does* have a flagged packet at the outset of $\mathtt{t}$, we argue that it will have height *at least* O when **Send Packet** is called in round $\mathtt{t}$ as follows. Let $\mathtt{t}_0 < \mathtt{t}$ denote the round $O_{N_1,R}$ first sent (a copy of) the packet to R. We first show that N_1 will *not* get confirmation of receipt from R (as in Definition 7.6) for the packet at any point between rounds $\mathtt{t}_0$ and $\mathtt{t}-1$ (inclusive). To see this, note that since we are Case 2, R has not accepted the flagged packet by the start of $\mathtt{t}$. This means that at all times between $\mathtt{t}_0$ and the start of $\mathtt{t}$, $RR < \mathtt{t}_0$[20]. Meanwhile, by Statement 3 of Lemma 7.7, $FR = \mathtt{t}_0$ and $H_{FP} \neq \perp$ at the start of $\mathtt{t}$. Since these do not change values before *Reset Outgoing Variables* is called in round $\mathtt{t}$, line (**6.34**) guarantees that if $H_{FP} < O$, then line (**6.35**) will be reached, and thus in either case $H_{FP} \geq O$ after the call to *Reset Outgoing Variables*.

Therefore, since R will necessarily receive and accept the flagged packet sent (by the same argument used in Case 1), we may apply Lemma 7.14 to argue that $\phi \leq -O$, which is (7) (for $l = 2$).

INDUCTION STEP. Assume the lemma is true for any chain of length less that or equal to $l-1$, and let $\mathcal{C}$ be a chain of length l ($l > 2$). Since we will be applying the induction hypothesis, we extend and

[20]By Statement 3 of Claim 7.7, the packet flagged in $\mathtt{t}_0$ is the only packet $O_{N_1,R}$ can send to R between $\mathtt{t}_0+1$ and the time R receives this flagged packet. Since we know R has still not accepted this flagged packet by the outset of $\mathtt{t}$, this means that between $\mathtt{t}_0$ and $\mathtt{t}-1$, RR cannot be changed as on (**6.53**). Since RR begins each transmission equal to -1 ((**3.31**) and (**6.64**)) and can only be changed after this on (**6.53**), necessarily $RR < \mathtt{t}_0$ through the start of $\mathtt{t}$.

change our notation as follows: Let O_{N_i,N_j} (respectively I_{N_i,N_j}) denote the height of N_i's outgoing (respectively incoming) buffer along edge $E(N_i, N_j)$ *at the start of round* `t` (before, the notation referred to the *buffer*, now it will refer to the buffer's *height*). Notice that if $O_{N_1,N_2} \leq I_{N_2,N_1}$, then:

$$\phi \leq -O_{N_2,N_3} + (l-1) - 1 \leq -I_{N_2,N_1} + l - 2 \leq -O_{N_1,N_2} + l - 2 \tag{8}$$

where the first inequality is from the induction hypothesis applied to the chain $N_2 \dots R$, the second follows from Lemma 6.3, and the third follows from the fact we are assuming $O_{N_1,N_2} \leq I_{N_2,N_1}$. Therefore, both (6) and (7) are satisfied. We may therefore assume in both cases below:

$$O_{N_1,N_2} > I_{N_2,N_1} \tag{9}$$

Case 1: O_{N_1,N_2} had a flagged packet at the start of `t` that was already accepted by N_2. If $O_{N_1,N_2} = I_{N_2,N_1} + 1$, then by the same string of inequalities as in (8), we would have $\phi \leq -O_{N_1,N_2} + l - 1$, which is (6). Therefore, it remains to consider the case:

$$O_{N_1,N_2} \geq I_{N_2,N_1} + 2 \tag{10}$$

By an analogous argument to the one made in the BASE CASE, a packet will be transfered and accepted across $E(N_1, N_2)$ in round `t` that will cause the non-duplicated potential to change by an amount less than or equal to:

$$(-O_{N_1,N_2} + 1) + I_{N_2,N_1} + 1 \tag{11}$$

Also, when the receiving node N_2 accepts this packet as on (**6.53**), the height of the corresponding buffer increases by one on this line. We emphasize this fact for use below:

> **Fact:** After the Routing Phase but before the call to *Re-Shuffle* in round `t`, N_2's incoming buffer along $E(N_1, N_2)$ has height $I_{N_2,N_1} + 1$.

Meanwhile, we may apply the induction hypothesis to the chain $\mathcal{C}' := N_2 \dots R$, so that the change in non-duplicated potential due to contributions 1 and 2 (in the hypothesis of the Lemma) on $\mathcal{C}'$ is less than or equal to:

(a) $-O_{N_2,N_3} + (l-1) - 1$, if O_{N_2,N_3} had a flagged packet at the start of `t` that was already accepted by N_3.

(b) $-O_{N_2,N_3} + (l-1) - 2$, otherwise.

Adding these contributions to (11), we have that:

$$\begin{aligned} \phi &\leq ((-O_{N_1,N_2} + 1) + I_{N_2,N_1} + 1) + (-O_{N_2,N_3} + (l-1) - x) \\ &= (-O_{N_1,N_2} + l - 1) + (-O_{N_2,N_3} + I_{N_2,N_1}) + (2 - x), \end{aligned} \tag{12}$$

where $x = 1$ or 2, depending on whether we are in case (a) or (b) above. By Lemma 6.3, $-O_{N_2,N_3} + I_{N_2,N_1}$ is either 0 or -1. If $-O_{N_2,N_3} + I_{N_2,N_1} = -1$, then $(-O_{N_2,N_3} + I_{N_2,N_1}) + (2 - x) \leq 0$, regardless whether $x = 1$ or 2, and hence (12) implies (6). Also, if $x = 2$, then $(-O_{N_2,N_3} + I_{N_2,N_1}) + (2 - x) \leq 0$ (by Lemma 6.3), and hence hence (12) implies (6). It remains to consider the case $x = 1$ and $-O_{N_2,N_3} + I_{N_2,N_1} = 0$, in which case (12) becomes:

$$\phi \leq (-O_{N_1,N_2} + l - 1) + 1 \tag{13}$$

In order to obtain (6) from (13), we therefore need to account for a drop of at least one more to ϕ. We will obtain this by the second contribution to ϕ (see statement of Lemma) by arguing:

(a) After the Routing Phase of round `t` but before the call to *Re-Shuffling*, the *fullest* buffer of N_2 has height $O_{N_2,N_3}+1$, and there is at least one *incoming* buffer of N_2 that has this height. In particular, during the call to *Re-Shuffle* in round `t`, the first buffer chosen to transfer a packet *from* will be an incoming buffer of height $O_{N_2,N_3}+1$.

(b) After the Routing Phase of round `t` but before the call to *Re-Shuffling*, the *emptiest* buffer of N_2 has height $O_{N_2,N_3}-1$, and there is at least one *outgoing* buffer of N_2 that has this height. In particular, during the call to *Re-Shuffle* in round `t`, the first buffer chosen to transfer a packet *to* will be an outgoing buffer of height $O_{N_2,N_3}-1$.

Notice that if I can show these two things, this case will be done, as during the first call to *Re-Shuffle* in round `t`, we will have $M-m \geq (O_{N_2,N_3}+1)-(O_{N_2,N_3}-1) \geq 2$ (the call to *Adjust Heights* can only help this inequality since the selection process on (**7.72-73**) and the two items above guarantee (**7.80**) and (**7.86**) will both fail if reached), and consequently the re-shuffle on (**7.89-90**) will cause a drop of at least one to ϕ.

We first argue (a). As noted at the beginning of Case 1 of the INDUCTION STEP, Fact 1 implies that there will exist an incoming buffer of the required height (since we are assuming $O_{N_2,N_3}=I_{N_2,N_1}$). Also, at the start of `t`, since N_2 has an outgoing buffer of height O_{N_2,N_3} (namely, the outgoing buffer along $E(N_2,N_3)$), Lemma 6.3 guarantees that all of N_2's incoming buffers have hieght at most O_{N_2,N_3} at the start of `t`; and also that all of N_2's outgoing buffers have height at most $O_{N_2,N_3}+1$ at the start of `t`. During the Routing Phase but before the Re-Shuffle Phase of `t`, outgoing buffers cannot *increase* in height (**6.33**) and incoming buffers cannot increase in height by more than one (**6.53**). Therefore, after transferring packets but before Re-Shuffling in round `t`, the fullest buffer in N_2 has height at most $O_{N_2,N_3}+1$, and as already argued, at least one incoming buffer has this height. The last part of (a) is immediate from the selection rules in (**7.72**).

We now argue (b). Since $x=1$, we are in the case the outgoing buffer along $E(N_2,N_3)$ had a flagged packet at the start of `t` that had already been accepted by N_3 in some round $\mathtt{t}_0 < \mathtt{t}$. By a similar argument that was used in Case 1 of the BASE CASE, the outgoing buffer along $E(N_2,N_3)$ will reach lines (**6.32-33**) in round `t`. In particular, the height of the outgoing buffer along $E(N_2,N_3)$ will drop by one on (**6.33**), and thus this buffer has height $O_{N_2,N_3}-1$ after the call to *Reset Outgoing Variables*. Since this height cannot change before the call to *Re-Shuffle*, this outgoing buffer has height $O_{N_2,N_3}-1$ after the Routing Phase (but before the call to *Re-Shuffle*) in round `t`. Also, $O_{N_2,N_3}-1$ is a lower bound for the *emptiest* buffer in N_2 just before the call to *Re-Shuffle* in round `t`, argued as follows. At the start of `t`, since N_2 has an incoming buffer of height $I_{N_2,N_1}=O_{N_2,N_3}$ (namely, the incoming buffer along $E(N_1,N_2)$), Lemma 6.3 guarantees that all of N_2's incoming buffers have hieght at least $O_{N_2,N_3}-1$ at the start of `t`; and also that all of N_2's outgoing buffers have hieght at least O_{N_2,N_3} at the start of `t`. During the Routing Phase but before the Re-Shuffle Phase of `t`, incoming buffers cannot *decrease* in height (**6.53**) and outgoing buffers can decrease in height by at most one (**6.33**). Therefore, after transferring packets but before Re-Shuffling in round `t`, the emptiest buffer in N_2 has height at least $O_{N_2,N_3}-1$, and as already argued, at least one outgoing buffer has this height. The last part of (b) is immediate from the selection rules in (**7.73**).

Case 2: Either O_{N_1,N_2} has no flagged packet at the start of `t`, or if so, it has not yet been accepted by N_2. By the same argument[21] used in Case 2 of the BASE CASE, there will be a packet transferred across $E(N_1, N_2)$ and accepted by N_2 in round t, and this packet will have height at least O_{N_1,N_2} in N_1's outgoing buffer. Therefore, by Lemma 7.14, the change in non-duplicated potential due to this packet transfer is less than or equal to:

$$-O_{N_1,N_2} + I_{N_2,N_1} + 1 \tag{14}$$

Meanwhile, we may apply the induction hypothesis to the chain $\mathcal{C}' := N_2 \dots R$, so that the change in non-duplicated potential due to contributions 1 and 2 (in the hypothesis of the Lemma) on $\mathcal{C}'$ is less than or equal to:

(a) $-O_{N_2,N_3} + (l-1) - 1$, if O_{N_2,N_3} had a flagged packet at the start of t that was already accepted by N_3.

(b) $-O_{N_2,N_3} + (l-1) - 2$, otherwise.

Adding these contributions to (14), we have that:

$$\begin{aligned}\phi &\le (-O_{N_1,N_2} + I_{N_2,N_1} + 1) + (-O_{N_2,N_3} + (l-1) - x) \\ &= (-O_{N_1,N_2} + l - 2) + (-O_{N_2,N_3} + I_{N_2,N_1}) + (2 - x),\end{aligned} \tag{15}$$

where $x = 1$ or 2, depending on whether we are in case (a) or (b) above. Since the first term of (15) matches (7) and the latter two terms match the latter two terms of (12), we follow the argument of Case 1 above to conclude the proof. ■

8 Routing Against a (Node-Controlling+Edge-Scheduling) Adversary

8.1 Definitions and High-Level Description of the Protocol

In this section, we define the variables that appear in the next section and describe how they will be used.

As in the protocol for the edge-scheduling adversary model, the sender first converts the input stream of messages into codewords, and then transmits a single codeword at a time. The sender will allow (at most) $4D$ rounds for this codeword to reach the receiver (for the edge-scheduling protocol, we only allowed $3D$ rounds; the extra D rounds will be motivated below). We will call each attempt to transfer a codeword a *transmission*, usually denoted by T. 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*.

As mentioned in Section 2.2, in the absence of a node-controlling adversary, the only difference between the present protocol and the one presented in Section 4 is that digital signatures are used to authenticate the sender's packets and also accompany packet transfers for later use to identify corrupt nodes. 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

[21] For the argument in the BASE CASE, we used the fact that the receiver's incoming buffer had height zero in order to conclude $H_{OUT} > H_{IN}$ (and thus a packet would be sent). Here, we use instead (9) to come to the same conclusion.

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*.

We give now a brief description of how we handle transmission failures in each of the three cases:

F2. The receiver could not decode, and the sender has inserted less than 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

Below is a short description of the specific kind of information nodes will be required to sign and store when communicating with their neighbors, and how this information will be used to identify corrupt nodes in each case F2-F4.

> Case F2. Anytime a packet at height h in an outgoing buffer of A is transferred to an incoming buffer of B at height h', A's potential will drop by h and B's potential will increase by h'. So for directed edge $E(A, B)$, A and B will each need to keep two values, the cumulative decrease in A's potential from packets leaving A, and the cumulative increase in B's potential from those packets entering B. These quantities are updated every time a packet is transferred across the edge, along with a tag indicating the round index and a signature from the neighbor validating the quantities and round index. Loosely speaking, case F2 corresponds to *packet duplication*. If a corrupt node attempts to slow transmission by duplicating packets, that node will have introduced extra potential in the network that cannot be accounted for, and the signing of potential changes will allow us to identify such a node.
>
> Case F3. A and B will keep track of the net number of packets that have travelled across edge $E(A, B)$. This number is updated anytime a packet is passed across the edge, and the updated quantity, tagged with the round index, is signed by both nodes, who need only store the most recent quantity. Loosely speaking, case F3 corresponds to *packet deletion*. In particular, the information signed here will be used to find a node who input more packets than it output, and such that the node's capacity to store packets in its buffers cannot account for the difference.
>
> Case F4. For each packet p corresponding to the current codeword, A and B will keep track of the net number of times the packet p has travelled across edge $E(A, B)$. This quantity is updated every time p flows across the edge, and the updated quantity, tagged with the round index, is signed by both nodes, who for each packet p need only store the most recent quantity. We will show in Section 10 that whenever case F4 occurs, the receiver will have necessarily received a duplicated packet (corresponding to the current codeword). Therefore, the information signed here will allow the sender to track this duplicated packet, looking for a node that outputted the packet more times than it inputted the packet.

We will prove that whenever cases F2-F4 occur, if the sender has all of the relevant quantities specified above, then he will necessarily be able to identify a corrupt node. Notice that each case of failure requires each node to transfer back only *one* signed quantity for each of its edges, and so the sender only needs n status report packets from each node.

We will show that the maximum number of failed transmissions that can occur before a corrupt node is necessarily identified and eliminated is n.[22] Because there are fewer than n nodes that can be corrupted by the adversary, the cumulative number of failed transmissions is bounded by n^2. This provides us with our main theorem regarding efficiency, stated precisely in Theorem 8.1 in Section 8.3 (note that the additive term that appears there comes from the n^2 failed transmissions).

8.2 Detailed Description of the Node-Controlling + Edge-Scheduling Protocol

In this section we give a more thorough description of our routing protocol for the node-controlling adversary model. Formal pseudo-code can be found in Section 9.

Setup. As in the edge-scheduling protocol, the sender has a sequence of messages $\{m_1, m_2, \dots\}$ that he will divide into *codewords* $\{b_1, b_2, \dots\}$. However, we demand each message has size $M' = \frac{6\sigma(P-2k)n^3}{\lambda}$, so that codewords have size $C' = \frac{M'}{\sigma}$, and these are then divided into packets of size $P' = P - 2k$, which will allow packets to have enough room[23] to hold two signatures of size k. Notice that the number of packets per codeword is $D = \frac{C'}{P'} = \frac{6(P-2k)n^3}{(P-2k)\lambda} = \frac{6n^3}{\lambda}$, which matches the value of D for the edge-scheduling protocol. One of the signatures that packets will carry with them comes from the sender, who will authenticate every packet by signing it, and the packets will carry this signature until they are removed from the network by the receiver. We re-emphasize Fact 1 from the edge-scheduling protocol, which remains true with these new values:

> **Fact 1′.** If the receiver has obtained $D - 6n^3 = (1-\lambda)\left(\frac{6n^3}{\lambda}\right)$ distinct and un-altered packets from any codeword, he will be able to decode the codeword to obtain the corresponding message.

The primary difference between the protocol we present here and that presented in Section 4 is the need to maintain and transmit information that will allow the sender to identify corrupt nodes. To this end, as part of the Setup, each node will have additional buffers:

3. *Signature Buffers.* Each node has a signature buffer along each edge to keep track of (outgoing/incoming) information exchanged with its neighbor along that edge. The signature buffers will hold information corresponding to changes in the following values *for a single transmission*. The following considers A's signature buffer along directed edge $E(A, B)$: 1) The net number of packets passed across $E(A, B)$; 2) B's cumulative change in potential due to packet transfers across $E(A, B)$. Additionally, for codeword packets corresponding to the current transmission only, the signature buffers will hold: 3) For each packet p that A has seen, the net number of times p has passed across $E(A, B)$ during the current transmission.

   Each of the three items above, together with the current round index and transmission index, ahve been signed by B. Since only a single (value, signature) pair is required for items 1)-2), while item 3) requires a (value, signature) pair for each packet, each signature buffer will need to hold at most $2 + D$ (value, signature) pairs.

[22] As mentioned in Section 2.2, the sender can eliminate a corrupt node as soon as he has received the status reports from every non-blacklisted node. The reason we require up to n transmissions to guarantee the identification of a corrupt node is that it may take this long for the sender to have the complete information he needs.

[23] The network is equipped with some minimal *bandwidth*, by which we mean the the number of bits that can be transferred by an edge in a single round. We will divide codewords into blocks of this size and denote the size by P, which therefore simultaneously denotes the size of any packet and also the network bandwidth. As in the edge-scheduling protocol, we assume $P > O(k + \log n)$, so that P' is well-defined.

4. *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 A's broadcast buffer will be able to hold the following (signed) information: 1) One *end of transmission* parcel (described below); 2) Up to n different *start of transmission* parcels (described below); 3) A list of blacklisted nodes and the transmission they were blacklisted or removed from the blacklist; 4) For each $B \in G$, a list of nodes for which B has claimed knowledge of their status report; and 5) For each $B \in G$ (including $B = A$), A's broadcast buffer can hold the content of up to n slots of B's signature buffer. Additionally, the broadcast buffer will also keep track of the *edges* across which it has already passed broadcasted information.

5. *Data Buffer.* This keeps a list of 1) All nodes that have been identified as corrupt and eliminated from the network by the sender; 2) Currently blacklisted nodes; and 3) Each node $A \in G$ will keep track of all pairs (N_1, N_2) such that N_2 is on the blacklist, and N_1 claims to know N_2's complete status report.

The sender's buffers are the same as in the edge-scheduling protocol's Setup, with the addition of four more buffers:

- *Data Buffer.* Stores all necessary information from the status reports, as well as additional information it will need to identify corrupt nodes. Specifically, the buffer is able to hold: 1) Up to n status report parcels from each node; 2) For up to n transmissions, the reason for failure, including the label of a duplicated packet (if relevant); 3) For up to n failed transmissions, a *participating list* of up to n nodes that were not on the blacklist for at least one round of the failed transmission; 4) A list of eliminated nodes and of blacklisted nodes and the transmission they were blacklisted; 5) The same as items 1)-3) of an internal node's Signature Buffer; and 6) The same as item 3) of an internal node's data buffer.
- *Broadcast Buffer.* Holds up to $2n$ *start of transmission* parcels and the labels of up to $n-1$ nodes that should be removed from the blacklist.
- *Copy of Current Packets Buffer.* Maintains a copy of all the packets that are being sent in the current transmission (to be used any time a transmission fails and needs to be repeated).

The receiver's buffers are as in the Edge-Scheduling protocol's Setup, with the addition of a *Broadcast Buffer*, *Data Buffer*, and *Signature Buffers* which are identical to those of an internal node.

The rest of the Setup is as in the edge-scheduling model, with the added assumption that 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. As in the edge-scheduling protocol, rounds consist of two stages followed by re-shuffling packets locally. The main difference between the two protocols will be the addition of signatures to all information, as well as the need to transmit the broadcast information, namely the status reports and *start* and *end of transmission* broadcasts, which inform the nodes of blacklisted and eliminated nodes and request status reports in the case a transmission fails. The two stages of a round are divided as they were for the edge-scheduling protocol, with the same treatment of routing codeword packets (with the addition of signatures). However, we will also require that each round allows all edges the opportunity to transmit broadcast information (e.g. status report parcels). Therefore, for every directed edge and every round of a transmission, there are four main packet transfers, two in each direction: codeword packets, broadcast parcels, and signatures confirming receipt of each of these. The order of transmitting these is succinctly expressed below in Figure 8.

At the start of any transmission T, the sender will determine which codeword is to be sent (the next one in the case the previous transmission was successful, or the same one again in the case the previous transmission failed). He will fill his Copy Of Current Packets Buffer with a copy of all of the codeword packets (to be used in case the transmission fails), and then fill his outgoing buffers with packets corresponding to this codeword. All codeword packets are signed by the sender, and these signatures will remain with the packets as they travel through the network to the receiver.

Stage	A		B
1	$H_A :=$ Height of buffer along $E(A,B)$ Height of flagged p. (if there is one) Round prev. packet was sent Confirmation of rec. of broadcast info.	$\longrightarrow$	
		$\longleftarrow$	H_B:=Height 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: • "Enough" bdcst info has passed $E(A,B)$, AND • B is not on A's blacklist/eliminated, AND – $H_A > H_B$ OR – B didn't rec. prev. packet sent	$\longrightarrow$	Receive packet if: • "Enough" bdcst info has passed $E(A,B)$ AND • A is not on B's blacklist/eliminated
		$\longleftarrow$	Broadcast Information

Figure 8: *Description of Communication Exchange Along Directed Edge $E(A,B)$ During the Routing Phase of Some Round.*

To compliment Figure 8 above, we provide a breif description of the information that should be passed across directed edge $E(A,B)$ ($B \neq S$ and $A \neq R$, and $A, B \in G$, i.e. not eliminated) during some transmission T. The precise and complete description can be found in the pseudo-code of Section 9. We state once and for all that if a node ever receives inaccurate or mis-signed information, it will act as if no information was received at all (e.g. as if the edge had failed for that stage).

Stage 1. A will send the same information to B as in the edge-scheduling protocol (height, height of flagged packet, round packet was flagged). Also, if A received a valid broadcast buffer from B in Stage 2 of the previous round, then A will send B confirmation of this fact. Also, if A knows that B has crucial broadcast information A needs, A specifies the type of broadcast information he wants from B. Meanwhile, A should receive the seven items that B signed and sent (see below), updating his internal variables as in the edge-scheduling protocol and updating its signature buffer, *provided* B has given a valid[24] signature.

At the other end, B will send the following seven items to A: 1) the transmission index; 2) index of the current round; 3) current height; 4) index of the round B last received a

[24]Here, "valid" means that A agrees with all the values sent by B, and B's signature is verified.

packet from A; 5) the net change in packet transfers so far across $E(A,B)$ for the current transmission; 6) B's cumulative increase in potential due to packet transfers across $E(A,B)$ in the current transmission; and 7) if a packet p was sent and received in Stage 2, the net number of times p has been transferred across $E(A,B)$ for the current transmission. Meanwhile, B will also receive the information A sent.

Stage 2. A will send a packet to B under the same conditions as in the edge-scheduling protocol, with the additional conditions: 1) A has received the sender's *start of transmission* broadcast (see below), and this information has passed across $E(A,B)$ or $E(B,A)$; 2) A and B are not on (A's version of) the blacklist; 3) A does not have any *end of transmission* information not yet passed across $E(A,B)$ or $E(B,A)$; and 4) A does not have any *changes to blacklist* information yet to pass across $E(A,B)$ or $E(B,A)$. We emphasize these last two points: if A (including $A=S$) has any *start of transmission*, *end of transmission*, or *changes to blacklist* information in its broadcast buffer that it has not yet passed along edge $E(A,B)$, then it will not send a packet along this edge.

If A does send a packet, the "packet" A sends includes a signature on the following seven items[25]: 1) transmission index; 2) index of the current round; 3) the packet itself with sender's signature; 4) index of the round A first tried to send this packet to B; 5) *One plus* the net change in packet transfers so far across $E(A,B)$ for the current transmission; 6) A's cumulative decrease in potential due to packet transfers across $E(A,B)$ in the current transmission *including the potential drop due to the current packet being transferred*; and 7) *One plus* the net number of times the packet currently being transferred has been transferred across $E(A,B)$ for the current transmission.

Also, A should receive broadcast information (if B has something in its broadcast buffer not yet passed along $E(A,B)$) and update its broadcast buffer as described by the *Update Broadcast Buffer Rules* below.

At the other end, B will receive and store the packet sent by A as in the edge-scheduling protocol, updating his signature buffer appropriately, with the added conditions: 1) B has received the sender's *start of transmission* broadcast, and this information has been passed across $E(A,B)$ or $E(B,A)$; 2) The packet has a valid signature from S; 3) A and B are not on (B's version of) the blacklist; 4) B does not have any *end of transmission* information not yet passed across $E(A,B)$ or $E(B,A)$; 5) B does not have any *changes to blacklist* information yet to pass across $E(A,B)$ or $E(B,A)$; and 6) The signatures on the seven items included with the packet from A is "valid.[24]" Additionally, if there is anything in B's broadcast buffer that has not been transferred along $E(A,B)$ yet, then B will send one parcel of broadcast information chosen according to the priorities: 1) The receiver's *end of transmission* parcel; 2) One of the sender's *start of transmission* parcels; 3) Changes to the blacklist or a node to permanently eliminate; 4) The identity of a node N on B's blacklist for which B has complete knowledge of N's status report; 5) The most recent status report parcel A requested in Stage 1 of an earlier round; and 6) Arbitrary status report parcels.

[25]Recall that packets have room to hold two signatures. The first will be the sender's signature that accompanies the packet until the packet is removed by the receiver. The second signature is the one indicated here, and this signature will be replaced/overwritten by the sending node every time the packet is passed across an edge.

For any edge $E(A,S)$ or $E(R,B)$, only broadcast information is passed along these directed edges, and this is done as in the rules above, with the exceptions mentioned below in the *Update Broadcast Buffer Rules* for S and R. Additionally, any round in which the sender is unable to insert any packets, he will increase the number of blocked rounds in his data buffer. The sender will also keep track of the total number of packets inserted in the current transmission in his data buffer.

Re-Shuffle Rules. The Re-Shuffle rules are exactly as in the edge-scheduling protocol, with the exception that node's record the changes in non-duplicated potential caused by re-shuffling packets locally.

Update Broadcast Buffer Rules for Internal Nodes. Looking at Stage 2 of the Routing Rules above, we have that in every round and for every edge $E(A,B)$, each node B will have the opportunity to send and receive exactly one parcel of *broadcast* information in addition to a single codeword packet. The order in which a node's broadcast buffer information is transmitted is described above in the Routing Rules.

For any node $A \in \mathcal{P} \setminus \{R,S\}$, we now describe the rules for updating their broadcast buffer when they receive broadcast information. Assume that A has received broadcast information along one of its edges $E(A,B)$, and has verified that it has a valid signature. We describe how A will update its Broadcast Buffer, depending on the nature of the new information:

> The received information is the receiver's *end of transmission* broadcast (see below). In this case, A will first make sure the transmission index is for the current transmission, and if so, the information is added to A's broadcast buffer, and edge $E(A,B)$ is marked as having already transmitted this information.
>
> The received information is the sender's *start of transmission* broadcast (see below). This broadcast consists of a single parcel containing information about the previous transmission, followed by up to $2n-2$ additional parcels (describing blacklisted/eliminated nodes and labels of up to n previous transmissions that have failed). When A receives a parcel from the *start of transmission* broadcast, if A does not already have it stored in its broadcast buffer, it will add it, and edge $E(A,B)$ is marked as having already transmitted this information. Additionally, A will handle the parcels concerning blacklisted nodes as described below. Finally, when A has received every parcel in the *start of transmission* broadcast, it will also remove from its blacklist any node *not* implicated in this broadcast (i.e. this will count as "A receives information concerning a node to remove from the blacklist," see below), as well as clearing its signature buffers for the new transmission.
>
> The received information indicates a node N to eliminate. If the information is current, then A will add the new information to its broadcast buffer and mark edge $E(A,B)$ as already having passed this information. If N is not already on A's list of eliminated nodes EN, then A will add N to EN (in its data buffer), clear all of its incoming and outgoing buffers, its signature buffers, and its broadcast buffer (with the exception of *start of transmission* parcels).
>
> The received information concerns a node N to add to or remove from the blacklist. If the received information *did not* originate in the current transmission (as signed by the sender) or A has more recent blacklist information regarding N, then A ignores the new information. Otherwise, the information is added to A's broadcast buffer, and edge $E(A,B)$ is marked as

having already transmitted this information. Additionally, parcels that are now outdated in A's broadcast buffer are deleted (such as N's status report parcels or a parcel indicating a node $\widehat{N}$ had N's complete status report).

If $N = A$, then A adds n of its own status report parcels to its broadcast buffer, choosing these n parcels based on information from the relevant *start* and *end of transmission* parcels. A also will add his own signature to each of these parcels, so that they each one will carry two signatures back to the sender (A's signature and the relevant neighbor's signature).

The received information indicates B has some node N's complete status report. If N is on A's blacklist for the transmission B claims knowledge for, then A stores the fact that B has complete knowledge of N in its data buffer (A will later use this information when requesting a specific parcel from B).

The received information pertains to some node N's status report the sender has requested. A will first make sure that N is on its version of the blacklist. If so, the information is added to A's broadcast buffer, and edge $E(A, B)$ is marked as having already transmitted this information. If this completes A's knowledge of N's status report, A will add to its broadcast buffer the fact that it now knows all of N's missing status report.

At the end of a transmission, all nodes will clear their broadcast buffers *except* parcels concerning a blacklisted node's status report. All nodes also clear their version of the blacklist (it will be restored at the beginning of the next transmission).

Update Broadcast Buffer Rules for Sender and Receiver. The receiver has the same rules as internal nodes for updating its broadcast buffer, with the addition that when there are exactly n rounds left in any transmission[26], R will add to its broadcast buffer a single (signed) parcel that indicates the transmission index, whether or not he could decode the current codeword, and the label of a duplicate packet he received (if there was one). We will refer to this as the receiver's *end of transmission* parcel.

The rules for the sender updating its broadcast buffer are slightly more involved, as the sender will be the one determining which information it requires of each node, as well as managing the blacklisted and eliminated nodes. The below rules dictate how S will update his broadcast buffer and status buffer at the end of a transmission, or when the sender receives new (appropriately signed) broadcast information along $E(S, B)$.

A transmission T has just ended. Note that the sender will have necessarily received and stored the receiver's *end of transmission* broadcast by the end of the transmission (Lemma 11.19). In the case that the transmission was successful, S will clear his outgoing buffers and Copy of Old Packets Buffer, then re-fill them with codewords corresponding to the next message. If EN denotes the eliminated nodes and $\mathcal{B}_{\mathtt{T}}$ denotes the nodes on the sender's blacklist at the end of this transmission, then the sender will set $\Omega_{\mathtt{T}+1} = (|EN|, |\mathcal{B}_{\mathtt{T}}|, F, 0)$, where F

[26]We note that because there is always an active honest path between sender and receiver, and the receiver's final broadcast has top priority in terms of broadcast order, the sender will necessarily receive this broadcast by the end of the transmission. Alternatively, we could modify our protocol to add an extra n rounds to allow this broadcast to reach the sender. However, the exposition is easier without adding an extra n rounds, and we will show that wasting the final n rounds of a transmission by having the receiver determine if it can decode with n rounds still left is not important, as the n wasted rounds is insignificant compared to the $O(n^3)$ rounds per transmission.

denotes the number of failed transmissions that have taken place since the last corrupt node was eliminated. Finally, the sender will delete the information in his data buffer concerning the number of packets inserted and the number of blocked rounds for the just completed transmission.

In the case that the transmission failed, S will clear his outgoing buffers and then re-fill them using the Copy of Old Packets Buffer, while leaving the latter buffer unchanged. The sender will determine the reason the transmission failed (F2-F4), and add this fact along with the relevant information from his own signature buffers to the data buffer. For any node (not including S or R or eliminated nodes) not on the sender's blacklist, the sender will add the node to the *participating list* $\mathcal{P}_{\mathtt{T}}$ in his data buffer, and then add each of these nodes to the blacklist, recording that $\mathtt{T}$ was the most recent transmission that the node was blacklisted. Also, the sender will set:

$$\Omega_{\mathtt{T}+1} = \begin{cases} (|EN|, |\mathcal{B}_{\mathtt{T}}|, F, p) & \text{if the transmission failed and } p \text{ was included in} \\ & \quad \text{the receiver's } \textit{end of transmission} \text{ parcel} \\ (|EN|, |\mathcal{B}_{\mathtt{T}}|, F, 1) & \text{if the transmission failed and } S \text{ inserted } D \text{ packets} \\ (|EN|, |\mathcal{B}_{\mathtt{T}}|, F, 2) & \text{otherwise} \end{cases}$$

For both a failed transmission and a successful transmission, the sender will sign and add to his broadcast buffer the following parcels, which will comprise the sender's *Start of Transmission* (SOT) broadcast: $\Omega_{\mathtt{T}+1}$, a list of eliminated and blacklisted nodes, and the reason for failure of each of the last (up to $n-1$) failed transmissions since the last node was eliminated. Note that each parcel added to the broadcast buffer regarding a blacklisted node includes the index of the transmission for which the node was blacklisted, and all parcels added to the broadcast buffer include the index of the transmission about to start (as a timestamp) and are signed by S. Notice that the rules regarding priority of transferring broadcast information guarantee that $\Omega_{\mathtt{T}+1}$ will be the first parcel of the SOT that is received, and because it reveals the number of blacklisted and eliminated nodes and the number of failed transmissions to expect, as soon as each node receives $\Omega_{\mathtt{T}+1}$, they will know exactly how many more parcels remain in the SOT broadcast. Nodes will not be allowed to transfer any (codeword) packets until the SOT broadcast for the current transmission is received in its entirety.

The sender receives the receiver's *end of transmission* parcel for the current transmission. The sender will store this parcel in its data buffer.

The sender receives information along $E(S, B)$ indicating B has some node N's complete status report. If N is on the sender's blacklist for the transmission B claims knowledge for, then the sender stores the fact that B has complete knowledge of N in its data buffer (the sender will later use this information when requesting a specific parcel from B).

The sender receives information along $E(S, B)$ that pertains to some node N's status report that the sender has requested. The sender will first make sure that N is on its blacklist and that the parcel received contains the appropriate information, i.e. the sender checks its data buffer to see which transmission N was added to the blacklist and the reason this transmission failed, and makes sure the status report parcel is from this transmission and contains the information corresponding to this reason for failure. If the parcel has faulty information that has been signed by N, i.e. N sent back information that was not requested by the sender, then N is eliminated from the network. Otherwise, the sender will add the information to its data buffer. If the information completes N's missing status report, the sender updates his broadcast buffer indicating N's removal from the blacklist, including the index of the current transmission and his signature.

The sender will then determine if he has enough information to eliminate a corrupt node N'. If so, N' will be added to his list of *Eliminate Nodes*, and his broadcast buffer and data buffer will be wiped completely (*except* for the list of eliminated nodes). Also, he will abandon the current transmission and begin a new one corresponding to the same codeword. In particular, he will clear his outgoing buffers and re-fill them with the codewords in his Copy of Old Packets Buffer, leaving the latter unchanged, and he will skip to the "A transmission has just ended" case above, setting the *start of transmission* parcel $\Omega_{\mathrm{T}+1} = (|EN|, 0, 0, 0)$.

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

We state our results concerning the correctness, throughput, and memory of our adversarial routing protocol, leaving the analysis and proofs to Section 10.

Theorem 8.1. *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.*

Theorem 8.2. *The memory required of each node is at most $O(n^4(k + \log n))$.*

Proofs. See Section 10. ■

9 Pseudo-Code for Node-Controlling + Edge-Scheduling Protocol

We now modify the pseudo-code from our edge-scheduling adversarial protocol to pseudo-code for the (node-controlling + edge-scheduling) adversarial model. The two codes will be very similar, with differences emphasized by marking the line number in **bold**. The Re-Shuffle Rules will remain the same as in the edge-scheduling protocol, with the addition of line (**7.76**) (see Figure 7).

Setup

DEFINITION OF VARIABLES:

01 $n :=$ Number of nodes in G;
02 $D := \frac{3n^3}{\lambda}$;
03 $\mathtt{T} :=$ Transmission index;
04 $\mathtt{t} :=$ Stage/Round index;
05 $k :=$ Security Parameter;
06 $P :=$ Capacity of edge$= O(k + \log n)$;
07 **for every** $N \in \mathcal{P} \setminus S$
08 $BB \in [n^2 + 5n] \times \{0,1\}^{P+n}$; ## Broadcast Buffer
09 $DB \in [1..n^2] \times \{0,1\}^P$; ## Data Buffer. Holds BL and EN below, and info. as on line 151
10 $BL \in [1..n-1] \times \{0,1\}^P$; ## Blacklist
11 $EN \in [1..n-1] \times \{0,1\}^P$; ## List of Eliminated Nodes
12 $SIG_{N,N} \in \{0,1\}^{O(\log n)}$; ## Holds change in potential due to local re-shuffling of packets
13 **for every** $N \in G$
14 $SK, \{PK\}_i^n$ ## Secret Key for signing, Public Keys to verify sig's of all nodes
15 **for every** *outgoing* **edge** $E(N,B) \in G, B \neq S$ and $N \neq R$
16 $\mathsf{OUT} \in [2n] \times \{0,1\}^P$; ## Outgoing Buffer able to hold $2n$ packets
17 $SIG_{N,B} \in [D+3] \times \{0,1\}^{O(\log n)}$; ## Signature Buffer for current trans., indexed as follows:
$SIG[1]=$ net no. of *current codeword* p's transferred across $E(N,B)$
$SIG[2]=$ net change in B's pot. due to p. transfers across $E(N,B)$
$SIG[3]=$ net change in N's pot. due to p. transfers across $E(N,B)$
$SIG[p]=$ net no. of times packet p transferred across $E(N,B)$
18 $\tilde{p} \in \{0,1\}^P \cup \perp$; ## Copy of packet to be sent
19 $sb \in \{0,1\}$; ## Status bit
20 $d \in \{0,1\}$; ## Bit indicating if a packet was sent in prev. round
21 $FR \in [0..8D] \cup \perp$; ## Flagged Round (index of round N first tried to send $\tilde{p}$ to B)
22 $RR \in [-1..8D] \cup \perp$; ## Round Received (index of round that N last rec. a p. from A)
23 $H \in [0..2n]$; ## Height of OUT. Also denoted H_{OUT} when there's ambiguity
24 $H_{FP} \in [1..2n] \cup \perp$; ## Height of Flagged Packet
25 $H_{IN} \in [0..2n] \cup \perp$; ## Height of incoming buffer of B
26 **for every** *outgoing* **edge** $E(N,B) \in G$, *including* $B = S$ and $N = R$
27 $bp \in \{0,1\}^P$; ## Broadcast Parcel received along this edge
28 $\alpha \in \{0,1\}^P$; ## Broadcast Parcel request
29 $c_{bp} \in \{0,1\}$; ## Verification bit of broadcast parcel receipt
30 **for every** *incoming* **edge** $E(A,N) \in G, A \neq R$ and $N \neq S$
31 $\mathsf{IN} \in [2n] \times \{0,1\}^P$; ## Incoming Buffer able to hold $2n$ packets
32 $SIG_{A,N} \in [D+3] \times \{0,1\}^{O(\log n)}$; ## Signature Buffer, indexed as on line 17
33 $p \in \{0,1\}^P \cup \perp$; ## Packet just received
34 $sb \in \{0,1\}$; ## Status bit
35 $RR \in \{0,1\}^{8D}$; ## Round Received index
36 $H \in [0..2n]$; ## Height of IN. Also denoted H_{IN} when there's ambiguity
37 $H_{GP} \in [1..2n] \cup \perp$; ## Height of Ghost Packet
38 $H_{OUT} \in [0..2n] \cup \perp$; ## Height of outgoing buffer or height of Flagged Packet of A
39 $sb_{OUT} \in \{0,1\}$; ## Status Bit of outgoing buffer of A
40 $FR \in \{0,1\}^{8D} \cup \perp$; ## Flagged Round index (from adjacent outgoing buffer A)
41 **for every** *incoming* **edge** $E(A,N) \in G$, *including* $A = R$ and $N = S$
42 $bp \in \{0,1\}^P$; ## Broadcast Parcel to send along this edge
43 $c_{bp} \in \{0,1\}$; ## Verification bit of packet broadcast parcel receipt

Figure 9: Pseudo-Code for Internal Nodes' Setup for the (Node-Controlling + Edge-Scheduling) Protocol

INITIALIZATION OF VARIABLES:

for every $N \in G$
  Receive Keys; ## Receive $\{PK\}_i^n$ and SK from KEYGEN
  Initialize BB, DB, BL, EN, $SIG_{N,N}$; ## Set $SIG_{N,N} = 0$, set each entry of DB and BB to $\perp$
  for every *incoming* **edge** $E(A,N) \in G, A \neq R$ and $N \neq S$
    Initialize IN, SIG; ## Set each entry in IN to $\perp$ and each entry of SIG to zero
    $p, H_{GP}, FR = \perp$;
    $sb, sb_{OUT}, c_p, H, H_{OUT} = 0$; $RR = -1$;
  for every *incoming* **edge** $E(A,N) \in G$, *including* $A = R$ and $N = S$
    $bp = \perp$; $c_{bp} = 0$;
  for every *outgoing* **edge** $E(N,B) \in G, B \neq S$ and $N \neq R$
    Initialize OUT, SIG; ## Set each entry in OUT to $\perp$ and each entry of SIG to zero
    $\tilde{p}, H_{FP}, FR, RR = \perp$;
    $sb, d, H, H_{IN}, 0$;
  for every *outgoing* **edge** $E(N,B) \in G$, *including* $B = S$ and $N = R$
    $bp, \alpha = \perp$; $c_{bp} = 0$;

Sender's Additional Setup

DEFINITION OF ADDITIONAL VARIABLES FOR SENDER:

$\mathcal{M} := \{m_1, m_2, \dots\}$ = Input Stream of Messages;
$COPY \in [D] \times \{0,1\}^P$:= Copy of Packets for Current Codeword;
$BB \in [3n] \times \{0,1\}^P$:= Broadcast Buffer;
$DB \in [1..n^3 + n^2 + n] \times \{0,1\}^P$:= Data Buffer, which includes:
  $BL \in [1..n] \times \{0,1\}^P$:= Blacklist;
  $EN \in [1..n] \times \{0,1\}^P$:= List of Eliminated Nodes;
$\kappa \in [0..D]$:= Number of packets corresponding to current codeword the sender has *knowingly* inserted;
$\Omega_{\mathtt{T}} \in \{0,1\}^{O(\log n)}$:= First parcel of Start of Transmission broadcast for transmission $\mathtt{T}$;
$\beta_{\mathtt{T}} \in [0..4D]$:= Number of rounds blocked in current transmission;
$F \in [0..n-1]$:= Number of failed transmissions since the last corrupt node was eliminated;
$\mathcal{P}_{\mathtt{T}} \in \{0,1\}^n$:= Participating List for current transmission;

INITIALIZATION OF SENDER'S VARIABLES:

$\kappa = 0$;
$\beta_1, F = 0$;
$\Omega_1 = (0,0,0,0)$;
Initialize BB, DB, $\mathcal{P}_1$; ## Set each entry of DB to $\perp$, add Ω_1 to BB, and set $\mathcal{P}_1 = G$
Distribute Packets;

Receiver's Additional Setup

DEFINITION OF ADDITIONAL VARIABLES FOR RECEIVER:

$I_R \in [D] \times (\{0,1\}^P \cup \perp)$:= Storage Buffer to hold packets corresponding to current codeword;
$\kappa \in [0..D]$:= Number of packets received corresponding to current codeword;
$\Theta_{\mathtt{T}} \in \{0,1\}^{O(k+\log n)}$:= End of Transmission broadcast for transmission $\mathtt{T}$;

INITIALIZATION OF RECEIVER'S VARIABLES:

$\kappa = 0$;
$\Theta_1 = \perp$;
for every *outgoing* **edge** $E(R,B) \in G$:
  $bp, \alpha = \perp$;
Initialize I_R; ## Sets each element of I_R to $\perp$

End Setup

Figure 10: Additional Setup Code for (Node-Controlling + Edge-Scheduling) Protocol

Transmission $\mathtt{T}$

01 **for every** $N \in G,\ N \notin EN$:
02 **for every** $\mathtt{t} < 2 * (4D)$ ## The factor of 2 is for the 2 stages per round
03 **if** $\mathtt{t} \pmod 2 = 0$ **then:** ## **STAGE 1**
04 ***Update Broadcast Buffer One***;
05 **for every** *outgoing* **edge** $E(N,B) \in G, N \neq R, B \neq S$
06 **if** $H_{FP} \neq \perp$: send $(H, \perp, \perp)$; **else:** send $(H-1, H_{FP}, FR)$;
07 receive *Signed*$(\mathtt{T}, \mathtt{t}, H_{IN}, RR, SIG[1], SIG[2], SIG[p])$; ## $SIG[3]$, 6^{th} coord sent on line 11, is kept as $SIG[2]$
08 ***Verify Signature Two***:
09 ***Reset Outgoing Variables***;
10 **for every** *incoming* **edge** $E(A,N) \in G, N \neq S, A \neq R$ ## "p" on line 11 refers to last p. rec'd on $E(A,N)$
11 send *Sign*$(\mathtt{T}, \mathtt{t}, H, RR, SIG[1], SIG[3], SIG[p])$; ## If p was from an *old* codeword, send instead:
Sign$(\mathtt{T}, \mathtt{t}, H, RR, SIG[1], SIG[3], \perp)$
12 $sb_{OUT} = 0$; $FR = \perp$;
13 receive $(H, \perp, \perp)$ **or** (H, H_{FP}, FR); ## If $H = \perp$ or $FR{>}RR$, set $sb_{OUT}{=}1$; and
$H_{OUT}{=}H_{FP}$; O.W. set $H_{OUT}{=}H$; $sb_{OUT}{=}0$;
14 **else if** $\mathtt{t} \pmod 2 = 1$ **then:** ## **STAGE 2**
15 ***Send/Receive Broadcast Parcels***;
16 **for every** *outgoing* **edge** $E(N,B) \in G, N \neq R, B \neq S$
17 **if** $H_{IN} \neq \perp$ **then:**
18 ***Create Flagged Packet***;
19 **if** $sb{=}1$ **or** ($sb{=}0$ **and** $H > H_{IN}$) **then:**
20 ***Send Packet***;
21 **for every** *incoming* **edge** $E(A,N) \in G, N \neq S, A \neq R$
22 ***Receive Packet***;

23 **if** $N \notin \{S, R\}$ **and** N has rec'd SOT broadcast for $\mathtt{T}$ **then:** ***Re-Shuffle***;
24 **else if** $N = R$ **and** N has rec'd SOT broadcast for $\mathtt{T}$ **then:** ***Receiver Re-Shuffle***;
25 **else if** $N = S$ **then:**
26 ***Sender Re-Shuffle***;
27 **if** *All (non-*$\perp$*) values* S *received on line 07 had* $H_{IN} = 2n$ **then:** $\beta_{\mathtt{T}} = \beta_{\mathtt{T}} + 1$;

28 **if** $\mathtt{t} = 2(4D - n)$ **and** $N = R$ **then:** ***Send End of Transmission Parcel***;
29 **if** $\mathtt{t} = 2(4D)$ **and** $N = S$ **then:** ***Prepare Start of Transmission Broadcast***;
30 **if** $\mathtt{t} = 2(4D)$ **then:** ***End of Transmission Adjustments***;

End Transmission $\mathtt{T}$

31 ***Okay to Send Packet***
32 **if**
- N does not have $(\Omega_{\mathtt{T}}, \mathtt{T})$ in BB OR
- N has $(\Omega_{\mathtt{T}}, \mathtt{T})$ with $\Omega_{\mathtt{T}} = (|EN|, |\mathcal{B}_{\mathtt{T}}|, F, *)$, but has not yet rec'd $|EN|$ parcels as in line 200b, F parcels as in line 200c, or $|\mathcal{B}_{\mathtt{T}}|$ parcels as in line 200d OR
- N has rec'd the complete SOT broadcast, but every parcel hasn't yet passed across $E(N,B)$ OR
- N or $B \in BL$ OR
- N has $\Theta_{\mathtt{T}} \in BB$, but this has not passed across $E(N,B)$ yet OR
- N has BL info. in BB (as on line 115, items 3 or 4) not yet passed across $E(N,B)$

33 Return False;
34 **else:** Return True;

35 ***Okay to Receive Packet***
36 **if**
- N does not have $(\Omega_{\mathtt{T}}, \mathtt{T})$ in BB OR
- N has $(\Omega_{\mathtt{T}}, \mathtt{T})$ with $\Omega_{\mathtt{T}} = (|EN|, |\mathcal{B}_{\mathtt{T}}|, F, *)$, but has not yet rec'd $|EN|$ parcels as in line 200b, F parcels as in line 200c, or $|\mathcal{B}_{\mathtt{T}}|$ parcels as in line 200d OR
- N has rec'd the complete SOT broadcast, but every parcel hasn't yet passed across $E(A,N)$ OR
- N or $A \in BL$ OR
- N has $\Theta_{\mathtt{T}} \in BB$, but this has not passed across $E(A,N)$ yet OR
- N has BL info. in BB (as on line 115, items 3 or 4) not yet passed across $E(A,N)$ OR

37 Return False;
38 **else:** Return True;

Figure 11: Routing Rules for Transmission $\mathtt{T}$, (Node-Controlling + Edge-Scheduling) Protocol

39 ***Reset Outgoing Variables***
40 $c_{bp} = 0$;
41 **if** $d = 1$: ## N sent a packet previous round
42 $d = 0$;
43 **if** $RR = \perp$ **or** $\perp \neq FR > RR$ ## Didn't receive conf. of packet receipt
44 $sb = 1$;
45 **if** $RR \neq \perp$:
46 **if** $\perp \neq FR \leq RR$: ## B rec'd most recently sent packet
47 **if** $N = S$ **then:** $\kappa = \kappa + 1$;
48 For $i = 1, 2, p$: $SIG[i]$ = value rec'd on line 07;
49 $SIG[3] = SIG[3] + H_{FP}$; ## If $N = S$, skip this line
50 $\mathsf{OUT}[H_{FP}] = \perp$; *Fill Gap*; ## Remove $\tilde{p}$ from OUT, shifting down packets on top ## of $\tilde{p}$ (if necessary) and adjusting $SIG_{N,N}$ accordingly
51 $FR, \tilde{p}, H_{FP} = \perp$; $sb = 0$; $H = H - 1$;
52 **if** $\perp \neq RR < FR$ **and** $\perp \neq H_{FP} < H$: ## B did *not* receive most recently sent packet
53 *Elevate Flagged Packet*; ## Swap packets in $\mathsf{OUT}[H]$ and $\mathsf{OUT}[H_{FP}]$; Set H_{FP}=H;

54 ***Create Flagged Packet***
55 **if** $sb = 0$ **and** $H > H_{IN}$: ## Normal Status, will send top packet
56 $\tilde{p} = \mathsf{OUT}[H]$; $H_{FP} = H$; $FR = \mathtt{t}$;

57 ***Send Packet***
58 $d = 1$;
59 **if *Okay to Send Packet* then:** ## If $\tilde{p}$ is from an *old* codeword, send instead:
60 send $Sign(\mathtt{T}, \mathtt{t}, \tilde{p}, FR, SIG[1]{+}1, SIG[3] + H_{FP}, SIG[\tilde{p}]{+}1)$; ## $Sign(\mathtt{T}, \mathtt{t}, \tilde{p}, FR, SIG[1], SIG[3] + H_{FP}, \perp)$

61 ***Receive Packet***
62 receive $Sign(\mathtt{T}, \mathtt{t} - 2, p, FR, SIG[1], SIG[2], SIG[p])$; ## $SIG[3]$, 6^{th} coord. sent on line 60, is kept as $SIG[2]$
63 **if** $H_{OUT} = \perp$ **or *Okay to Receive Packet*** is *false*: ##Didn't rec. A's ht. info, or BB info prevents p. transfer
64 $sb = 1$;
65 **if** $H_{GP} > H$ **or** ($H_{GP} = \perp$ **and** $H < 2n$):
66 $H_{GP} = H + 1$;
67 **else if** $sb_{OUT} = 1$ **or** $H_{OUT} > H$: ## A packet should've been sent
68 ***Verify Signature One***;
69 **if** (***Verify Signature One*** returns false **or** $p = \perp$ **or** *p not properly signed by S*) **then:** ## Signature from A was not valid, or ## Packet wasn't rec'd. or wasn't signed by S
70 $sb = 1$;
71 **if** $H_{GP} > H$ **or** ($H_{GP} = \perp$ **and** $H < 2n$):
72 $H_{GP} = H + 1$;
73 **else if** $RR < FR$: ## Packet was rec'd and should keep it
74 For $i = 1, 2, p$: $SIG[i]$ = value rec'd on line 62;
75 $SIG[3] = SIG[3] + H_{GP}$; ## If $N = R$, skip this line
76 **if** $H_{GP} = \perp$: $H_{GP} = H + 1$; ## If no slot is saved for p, put it on top
77 $\mathsf{IN}[H_{GP}] = p$;
78 $sb = 0$; $H = H + 1$; $H_{GP} = \perp$; $RR = \mathtt{t}$;
79 **else:** ## Packet was rec'd, but already had it
80 $sb = 0$; *Fill Gap*; $H_{GP} = \perp$; ## See comment about *Fill Gap* on line 82 below
81 **else:** ## A packet should NOT have been sent
82 $sb = 0$; *Fill Gap*; $H_{GP} = \perp$; ## If packets occupied slots *above* the Ghost ## Packet, then *Fill Gap* will Slide them *down* one slot, ## updating $SIG_{N,N}$ to reflect this shift, if necessary

83 ***Verify Signature One***
84 **if** *Signature is Valid* **and** *Values are correct* ## N verifies the values A sent on line 60 are consistent:
85 Return *true*; ## Change in $SIG[1]$ and $SIG[p]$ is '1', change in $SIG[2]$ is
86 **else:** ## at least H_{GP}, $(\mathtt{T}, \mathtt{t})$ is correct and p. has sender's sig
87 Return *false*;

88 ***Verify Signature Two*** ## N verifies the values B sent on line 11 are consistent:
89 **if** *Signature is NOT Valid* **or** *Values are NOT Correct*: ## Change in $SIG[1]$ and $SIG[p]$ is '1', change in $SIG[2]$
90 $RR, H_{IN} = \perp$; ## is at most H_{FP}, and $\mathtt{T}$ and $\mathtt{t}$ are correct

Figure 12: Routing Rules for Transmission $\mathtt{T}$, (Node-Controlling + Edge-Scheduling) Protocol (cont)

91 ***Send/Receive Broadcast Parcels***
92   **for every** *outgoing* **edge** $E(N,B) \in G$, *including* $N = R, B = S$
93   receive bp;
94   ***Update Broadcast Buffer Two***;
95   **for every** *incoming* **edge** $E(A,N) \in G$, *including* $N = S, A = R$
96   ***Determine Broadcast Parcel to Send***;
97   send bp;

98 ***Update Broadcast Buffer One***
99   **for every** *outgoing* **edge** $E(N,B) \in G$, *including* $N = R, B = S$
100   **if** $bp \neq \perp$ **then:**
101   send c_{bp};
102   ***Broadcast Parcel to Request***;
103   send α;
104   **for every** *incoming* **edge** $E(A,N) \in G$, *including* $N = S, A = R$
105   receive c_{bp}; receive α;
106   **if** $\alpha \neq \perp$ **then:** Update Broadcast Buffer; ## Update BB to preferentially send α
107   **if** $c_{bp} = 1$ **then:** Update Broadcast Buffer; ## Update BB that bp crossed $E(A,N)$
108     $c_{bp} = 0$;

109 ***Update Broadcast Buffer Two***
110   **if** $\perp \neq bp$ has valid sig. **and** $\begin{cases} N \text{ has received full } SOT \text{ broadcast for } \mathtt{T} \quad \text{OR} \\ bp \text{ is a valid } SOT \text{ broadcast parcel rec'd in correct order (see 115 and 200)} \end{cases}$
      ## Here, a "valid" signature means both from B and the from node bp originated from, and
      ## a "valid" SOT parcel means that N has already received all SOT parcels that
      ## should have arrived before bp, as indicated by the ordering of line 115, items 2a-2d
111     $c_{bp} = 1$;
112   **if** $N = S$: ***Sender Update Broadcast Buffer***;
113   **else:** ***Internal Node and Receiver Update Broadcast Buffer***;

114 ***Determine Broadcast Parcel to Send***
115   Among all information in BB, choose some $bp \in BB$ that has not passed along $E(A,N)$ by priority:
    1) The receiver's *end of transmission* parcel $\Theta_{\mathtt{T}}$
    2) The sender's *start of transmission* (SOT) broadcast, in the order indicated on line 200:
      a) $(\Omega_{\mathtt{T}}, \mathtt{T})$ b) $(\widehat{N} \in EN, \mathtt{T})$ c) $(\mathtt{T}', Fi, \mathtt{T})$ d) $(\widehat{N} \in BL, \mathtt{T}', \mathtt{T})$
    3) $(\widehat{N}, 0, \mathtt{T})$ = label of a node to *remove* from the blacklist, see line 165
    4) $(N, \widehat{N}, \mathtt{T}')$ = label of a node $\widehat{N}$ on BL for which N has the complete status report for $\mathtt{T}'$, see line 155
    5) A status report parcel requested by A as indicated by α (received on line 105)
    6) An arbitrary status report parcel of a node on N's blacklist

116 ***Broadcast Parcel to Request***
117   $\alpha = \perp$;
118   **if** B is on N's blacklist and N is missing a status report from B:
119   Set α to indicate B's label and an index of the parcel N is missing from B;
120   **else if** DB indicates that B has complete status report for some node $\widehat{N}$ on BL (see lines 150-151, 155):
121   **if** N is missing a status report of node $\widehat{N}$:
122   Set α to the label of the node $\widehat{N}$ and the index of a status report parcel from $\widehat{N}$ that N is missing;

Figure 13: Routing Rules for Transmission $\mathtt{T}$, (Node-Controlling + Edge-Scheduling) Protocol (cont)

123 ***Internal Node and Receiver Update Broadcast Buffer***
Below, a broadcast parcel *bp* is "*Added*" only if it is not already in BB. Also, view BB as being
indexed by each *bp* with $n-1$ slots for each parcel to indicate which edges *bp* has already traversed.
Then when *bp* is removed from BB, the edge "markings" are removed as well.
124 **if** $bp = \Theta_{\mathtt{T}}$ is receiver's *end of transmission* parcel (for current transmission $\mathtt{T}$, see line 179):
125 Add *bp* to BB and mark edge $E(N,B)$ as having passed this info.;
126 **else if** $bp = (\Omega_{\mathtt{T}}, \mathtt{T})$ is the first parcel of the sender's *start of transmission* (SOT) broadcast (see line 200a):
127 Add *bp* to BB, and mark edge $E(N,B)$ as having passed this information;
128 **if** $\Omega_{\mathtt{T}} = (*, 0, *, *)$: Clear all entries of SIG, and set $SIG_{N,N} = 0$;
129 **else if** $bp = (\widehat{N}, \mathtt{T})$ is from the SOT broadcast indicating a node to *eliminate*, as on line 200b:
130 Add *bp* to BB and mark edge $E(N,B)$ as having passed this info.;
131 **if** $\widehat{N} \notin EN$: ## N is just learning $\widehat{N}$ is to be eliminated
132 Add $\widehat{N}$ to EN;
133 Clear all incoming and outgoing buffers, clear all entries of SIG, and set $SIG_{N,N} = 0$;
134 Clear BB, EXCEPT for parcels from current SOT broadcast; Clear DB, EXCEPT for EN;
135 **else if** $bp = (\mathtt{T}', Fi, \mathtt{T})$ is from the SOT broadcast indicating why a previous trans. failed, as on line 200c:
136 Add *bp* to BB and mark edge $E(N,B)$ as having passed this information;
137 **else if** $bp = (\widehat{N}, \mathtt{T}', \mathtt{T})$ is from the SOT broadcast indicating a node to blacklist, as on line 200d:
138 Add $\widehat{N}$ to BL; Add *bp* to BB and mark edge $E(N,B)$ as having passed this information;
139 Remove outdated info. from BB and DB;
This includes for any trans. $\mathtt{T}'' \neq \mathtt{T}'$ removing from DB all entries of form $(\widetilde{B}, \widehat{N}, \mathtt{T}'')$, see line 115, item 4;
and removing from BB: 1) $(N, \widehat{N}, \mathtt{T}'')$, see line 115 item 4, and 2) Any status report parcel of $\widehat{N}$ for $\mathtt{T}''$
140 **if** $\widehat{N} = N$ has not already added its own status report info. corresponding to $\mathtt{T}'$ to BB:
The following reasons for failure come from SOT. See lines 190, 193, and 196-197
The information added in each case will be referred to as the node's ***status report*** for transmission $\mathtt{T}'$
141 **if** entries of $SIG_{N,N}$ and SIG correspond to a transmission $\mathtt{T}'' \neq \mathtt{T}'$: *Clear* SIG and set $SIG_{N,N} = 0$;
142 **if** $\mathtt{T}'$ failed as in F2: For each incoming and outgoing edge, sign and add to BB: $(SIG[2], SIG[3], \mathtt{T}')$;
143 Also sign and add $(SIG_{N,N}, \mathtt{T}')$ to BB (see line 12 of Figure 9);
144 **else if** $\mathtt{T}'$ failed as in F3: For each incoming and outgoing edge, sign and add $(SIG[1], \mathtt{T}')$ to BB;
145 **else if** $\mathtt{T}'$ failed as in F4: For each incoming and outgoing edge, sign and add $(SIG[p], \mathtt{T}')$ to BB;
146 **if** N has received $|BL_{\mathtt{T}}|$ SOT parcels of form $(\widehat{N}, \mathtt{T}', \mathtt{T})$: Clear all entries of SIG and set $SIG_{N,N} = 0$;
147 **else if** $bp = (\widehat{N}, 0, \mathtt{T})$ is from sender, indicating a node to remove from BL, as on line 165:
148 Remove $\widehat{N}$ from BL; Add *bp* to BB and mark edge $E(N,B)$ as having passed this information;
149 Remove outdated info. from BB and DB as on line 139 above;
150 **else if** $bp = (B, \widehat{N}, \mathtt{T}')$ indicates B has a blacklisted node $\widehat{N}$'s complete status report for trans. $\mathtt{T}'$:
151 **if** $(\widehat{N}, \mathtt{T}', \mathtt{T})$ is on N's blacklist: Add fact that B has $\widehat{N}$'s complete status report to DB;
152 **else if** *bp* is a status report parcel for trans. $\mathtt{T}'$ of some node $(\widehat{N}, \mathtt{T}', \mathtt{T})$ on BL, see lines 140-145 and 200d:
153 **if** *bp* has valid sig. from $\widehat{N}$ and concerns correct info.:
N finds $(\widehat{N}, \mathtt{T}', \mathtt{T})$ and $(\mathtt{T}', Fi, \mathtt{T})$ in BB (from SOT broadcast) and checks that *bp* concerns correct info.
154 Add *bp* to BB, and mark edge $E(N,B)$ as having passed this information;
155 **if** *bp* completes N's knowledge of $\widehat{N}$'s missing status report for transmission $\mathtt{T}'$: Add $(N, \widehat{N}, \mathtt{T}')$ to BB;

156 ***Sender Update Broadcast Buffer*** ## Below, a parcel *bp* is "*Added*" only if it is not in DB
157 **if** $bp = \Theta_{\mathtt{T}}$ is receiver's *end of transmission* parcel (for current transmission $\mathtt{T}$):
158 Add *bp* to DB;
159 **else if** *bp* indicates B has a blacklisted node $\widehat{N}$'s complete status report for trans. $\mathtt{T}'$:
160 **if** $(\widehat{N}, \mathtt{T}', \mathtt{T})$ is on S's blacklist: Add $(B, \widehat{N}, \mathtt{T}')$ to DB;
161 **else if** *bp* is a status report parcel of some node $\widehat{N}$ on the sender's blacklist (see lines 140-145):
162 Add *bp* to DB;
163 **if** *bp* contains faulty info. but has a valid sig. from $\widehat{N}$: ***Eliminate*** $\widehat{N}$;
S checks DB for reason of failure and makes sure $\widehat{N}$ has returned an appropriate value
164 **if** *bp* completes the sender's knowledge of $\widehat{N}$'s missing status report from transmission $\mathtt{T}'$:
165 Sign $(\widehat{N}, 0, \mathtt{T})$ and add to BB; ## Indicates that $\widehat{N}$ should be removed from blacklist
166 Remove outdated info. from DB; Remove $(\widehat{N}, \mathtt{T}')$ from BL;
"Outdated" refers to parcels as on 159-160 whose second entry is $\widehat{N}$
167 **if** *bp* completes sender's knowledge of all relevant status reports from some transmission:
168 ***Eliminate*** $\widehat{N}$; ## S can eliminate a node. See pf. of Thm 8.1 for details

Figure 14: Routing Rules for Transmission $\mathtt{T}$, (Node-Controlling + Edge-Scheduling) Protocol (cont)

169 ***Eliminate*** $\widehat{N}$
170 *Add* $(\widehat{N}, \mathtt{T})$ to EN;
171 *Clear* BB, DB (except for EN), and signature buffers;
172 $\beta_{\mathtt{T}}, F = 0$;
173 $\mathcal{P}_{\mathtt{T}+1} = \mathcal{P} \setminus EN$;
174 $\Omega_{\mathtt{T}+1} = (|EN|, 0, 0, 0)$;
175 *Sign* and *Add* $\Omega_{\mathtt{T}+1}$ to BB;
176 **for every** $N \in EN$, *Sign* and *Add* $(N, \mathtt{T}+1)$ to BB;
177 *Halt* until ***End of Transmission Adjustments*** is called; ## S does not begin inserting p's until next trans., ## and S ignores all instructions for $\mathtt{T}$ until line 30

178 ***Send End of Transmission Parcel***
179 *Add signed* $\Theta_{\mathtt{T}} = (b, p', \mathtt{T})$ *to* BB ## b is a bit indicating if R could decode, p' is ## the label of a packet R rec'd twice, or else $\perp$

180 ***Prepare Start of Transmission Broadcast***
181 ## Let $\Theta_{\mathtt{T}} = (b, p', \mathtt{T})$ denote Sender's value obtained from Receiver's transmission above (as stored in DB)
182 **if** $b = 1$ **then:** ## R was able to decode
183 *Clear* each entry of signature buffers holding data corresponding to $\mathtt{T}$;
184 $\Omega_{\mathtt{T}+1} = (|EN|, |BL|, F, 0)$;
185 **else if** $b = 0$ **then:** ## R was *not* able to decode: a *failed* transmission
186 $F = F + 1$;
187 Set $\mathcal{P}_{\mathtt{T}} = \mathcal{P} \setminus (EN \cup BL)$ and add $(\mathcal{P}_{\mathtt{T}}, \mathtt{T})$ to DB;
188 For each $N \in \mathcal{P}_{\mathtt{T}} \setminus S$: Add $(N, \mathtt{T})$ to BL; ## $(N, \mathtt{T})$ records the trans. N was added to BL
189 *Clear* outgoing buffers;
190 **if** $p' \neq \perp$: ## R rec'd a duplicate packet
191 Add $(p', \mathtt{T})$ to DB; Add $SIG[p']$ to DB; ## Record that reason $\mathtt{T}$ failed was F4
192 $\Omega_{\mathtt{T}+1} = (|EN|, |BL|, F, p')$;
193 **else if** $\kappa < D$: ## S did not insert at least D packets
194 Add $(1, \mathtt{T})$ to DB; Add $SIG[2]$ and $SIG[3]$ to DB; ## Record that reason $\mathtt{T}$ failed was F2
195 $\Omega_{\mathtt{T}+1} = (|EN|, |BL|, F, 1)$;
196 **else:**
197 Add $(2, \mathtt{T})$ to DB; Add $SIG[1]$ to DB; ## Record that reason $\mathtt{T}$ failed was F3
198 $\Omega_{\mathtt{T}+1} = (|EN|, |BL|, F, 2)$;
199 *Clear* BB and $SIG[i]$ for each $i = 1, 2, p$; *Remove* $\Theta_{\mathtt{T}}$ from DB;
200 *Sign* and *Add* to BB: ## The *Start of Transmission (SOT)* broadcast
a) $(\Omega_{\mathtt{T}+1}, \mathtt{T}+1)$
b) For each $N \in EN$, add the parcel $(N, \mathtt{T}+1)$
c) For each failed transmission $\mathtt{T}'$ since the last node was eliminated, add the parcel $(\mathtt{T}', Fi, \mathtt{T}+1)$
Here, Fi is the reason trans. $\mathtt{T}'$ failed (F2, F3, or F4). See pf. of Thm. 8.1 for details
d) For each $N \in BL$, add the parcel $(N, \mathtt{T}', \mathtt{T}+1)$, where $\mathtt{T}'$ indicates the trans. N was last added to BL
201 $\beta_{\mathtt{T}} = 0$;

202 ***End of Transmission Adjustments***
203 **if** $N \neq S$: Clear $\Theta_{\mathtt{T}}$, BL, all parcels from *SOT* broadcast, and info. of form $(\widehat{N}, 0, \mathtt{T})$ from BB;
204 **for every** *outgoing* edge $E(N, B)$, $B \in G, N \neq R, B \neq S$:
205 **if** $H_{FP} \neq \perp$:
206 $OUT[H_{FP}] = \perp$; *Fill Gap*; ## Remove any flagged packet $\tilde{p}$ from OUT, shifting ## down packets on top of $\tilde{p}$ if necessary
207 $sb = 0$; $FR, H_{FP}, \tilde{p} = \perp$; $H = H - 1$;
208 **for every** *incoming* edge $E(A, N)$, $A \in G$, $A \neq R$:
209 $H_{GP} = \perp$; $sb = 0$; $RR = -1$; *Fill Gap*;
210 **if** $N = S$ **then:** ***Distribute Packets***;
211 **if** $N = R$ **then:** $\kappa = 0$; *Clear* I_R; ## Set each entry of I_R to $\perp$

212 ***Distribute Packets***
213 $\kappa = 0; H_{OUT} = 2n$; ## Set height of each outgoing buffer to $2n$
214 Fill each outgoing buffer with codeword packets;
If $\mathtt{T}$ was successful, make *new* codeword p's, and fill out. buffers and COPY with these.
If $\mathtt{T}$ failed or a node was just eliminated, use codeword packets in COPY to fill out. buffers.

Figure 15: Routing Rules for Transmission $\mathtt{T}$, (Node-Controlling + Edge-Scheduling) Protocol (cont)

10 Node-Controlling + Edge-Scheduling Protocol: Proofs of Theorems

We restate and prove our two main theorems for the node-controlling adversary Routing Protocol:

Theorem 8.2. *The memory required of each node is at most $O(n^4(k + \log n))$.*

Proof. By looking at the domains of the variables in Figures 9 and 10, it is clear that the Broadcast Buffer and Signature Buffer maintained by all nodes, and the Data Buffer of the sender and Storage Buffer of the receiver require the most memory. Each of these require $O(n^3(k + \log n))$ bits of memory, but each node must maintain $O(n)$ signature buffers, which yields the memory bound of $O(n^4(k+\log n))$. It remains to show that the domains described are accurate, i.e. that the protocol never calls for the nodes to store in more (or different) information than their domains allow. The proof of this fact walks through the pseudo-code and analyses every time information is added or deleted from each buffer, and it can be found in Section 11 (see Lemma 11.2). ∎

Theorem 8.1. *Except for the at most n^2 transmissions that may fail due to malicious activity, the routing protocol presented described in Sections 8 and 9 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.*

Theorem 8.1 will follow immediately from the following three theorems. As with the proofs for the edge-scheduling protocol, line numbers for the pseudo-code have form (**X.YY**), where **X** refers to the Figure number, and **YY** refers to the line number. It will be convenient to introduce new terminology:

Definition 10.1. We will say a node $N \in G$ *participated* in transmission $\mathtt{T}$ if there was at least one round in the transmission for which N was not on the (sender's) blacklist. The sender's variable that keeps track of nodes participating in transmission $\mathtt{T}$ (**10.69**) will be called the *participating list* for transmission $\mathtt{T}$, denoted by $\mathcal{P}_{\mathtt{T}}$ (it is updated at the end of every transmission on **15.187**).

Theorem 10.2. *Every transmission (regardless of its success/failure) lasts $O(n^3)$ rounds.*

Proof. Line (**11.02**) shows that each transmission, regardless of success or failure, lasts $4D = O(n^3)$ rounds. ∎

Theorem 10.3. *Suppose transmission $\mathtt{T}$ failed and at some later time (after transmission $\mathtt{T}$ but before any additional nodes have been eliminated) the sender has received all of the status report parcels from all nodes on $\mathcal{P}_{\mathtt{T}}$. Then the sender can eliminate a corrupt node.*

Proof. The proof of this theorem is rather involved, as it needs to address the three possible reasons (F2-F4) that a transmission can fail. It can be found in Section 10.1 below. ∎

Theorem 10.4. *After a corrupt node has been eliminated (or at the outset of the protocol) and before the next corrupt node is eliminated, there can be at most $n - 1$ failed transmissions $\{\mathtt{T}_1, \ldots, \mathtt{T}_{n-1}\}$ before there is necessarily some index $1 \leq i \leq n - 1$ such that the sender has the complete status report from every node on $\mathcal{P}_{\mathtt{T}_i}$.*

Proof. The theorem will follow from a simple observation:

> **Observation.** If $N \in \mathcal{P}_{\mathtt{T}}$, then the sender is not missing any status report parcel for N for any transmission prior to transmission $\mathtt{T}$. In other words, there is no transmission $\mathtt{T}' < \mathtt{T}$ such that N was blacklisted at the end of $\mathtt{T}'$ (as on **15.188**) and the sender is still missing status report information from N at the end of $\mathtt{T}$.
>
> **Proof.** Nodes are added to the blacklist whenever they were participating in a transmission that failed (**15.187-88**). Nodes are removed from the blacklist whenever the sender receives all of the status report information he requested of them (**14.164-166**), or when he has just eliminated a node (**15.171**), in which case the sender no longer needs status reports from nodes for old failed transmissions[27] (and in particular, this case falls outside the hypotheses of the Theorem). Since $\mathcal{P}_{\mathtt{T}}$ is defined as non-blacklisted nodes (**15.187**), the fact that $N \in \mathcal{P}_{\mathtt{T}}$ implies that N was not on the sender's blacklist at the end of $\mathtt{T}$. Also, notice the next line guarantees that *all* nodes not already on the sender's blacklist will be put on the blacklist if the transmission fails. Therefore, if N has not been blacklisted since the last node was eliminated (**15.169-177**), then there have not been any failed transmissions, and hence the sender is not missing any status reports. Otherwise, let $\mathtt{T}' < \mathtt{T}$ denote the last time N was put on the blacklist, as on (**15.188**). In order for N to be put on $\mathcal{P}_{\mathtt{T}}$ on line (**15.187**) of transmission $\mathtt{T}$, it must have been removed from the blacklist at some point between $\mathtt{T}'$ and the end of $\mathtt{T}$. In this case, the remarks at the start of the proof of this observation indicate the sender is not missing any status reports from N. ∎

Suppose now for the sake of contradiction that we have reached the end of transmission $\mathtt{T}_{n-1}$, which marks the $(n-1)^{st}$ transmission $\{\mathtt{T}_1, \dots, \mathtt{T}_{n-1}\}$ such that for each of these $n-1$ failed transmissions, the sender does not have the complete status report from at least one of the nodes that participated in the transmission. Define the set $\mathcal{S}$ to be the set of nodes that were necessarily *not* on $\mathcal{P}_{\mathtt{T}_{n-1}}$, and initialize this set to be empty.

Since the sender is missing some node's complete status report that participated in $\mathtt{T}_1$, there is some node $N_1 \in \mathcal{P}_{\mathtt{T}_1}$ from which the sender is still missing a status report parcel corresponding to $\mathtt{T}_1$ by the end of transmission $\mathtt{T}_{n-1}$. Notice by the observation above that N_1 will not be on $\mathcal{P}_{\mathtt{T}'}$ for any $\mathtt{T}_2 \le \mathtt{T}' \le \mathtt{T}_{n-1}$, so put N_1 into the set $\mathcal{S}$. Now looking at $\mathtt{T}_2$, there must be some node $N_2 \in \mathcal{P}_{\mathtt{T}_2}$ from which the sender is still missing a status report parcel from $\mathtt{T}_2$ by the end of transmission $\mathtt{T}_{n-1}$. Notice that $N_2 \neq N_1$ since $N_1 \notin \mathcal{P}_{\mathtt{T}_2}$, and also that $N_2 \notin \mathcal{P}_{\mathtt{T}_{n-1}}$ (both facts follow from the above observation), so put N_2 into $\mathcal{S}$. Continue in this manner, until we have found the $(n-2)^{th}$ distinct node that was put into $\mathcal{S}$ due to information the sender was still missing by the end of $\mathtt{T}_{n-2}$. But then $|\mathcal{S}| = n-2$, which implies that all nodes, except for the sender and the receiver, are not on $\mathcal{P}_{\mathtt{T}_{n-1}}$ (the sender and receiver participate in every transmission by Lemma 11.21). But now we have a contradiction, since Lemma 11.22 says that transmission $\mathtt{T}_{n-1}$ will not fail. ∎

We are now ready to prove Theorem 8.1, reserving the proof of Theorem 10.3 to the next section.

[27] The sender already received enough information to eliminate a node. Even though it is possible that other nodes acted maliciously and caused one of the failed transmissions, it is also possible that the node just eliminated caused all of the failed transmissions. Therefore, the protocol does not spend further resources attempting to detect another corrupt node, but rather starts anew with a reduced network (the eliminated node no longer legally participates), and will address future failed transmissions as they arise.

Proof of Theorem 8.1. By Theorem 10.2, every transmission lasts at most $O(n^3)$ rounds, so it remains to show that there are at most n^2 failed transmissions. By Theorem 10.4, by the end of at most $n-1$ failed transmissions, there will be at least one failed transmission $\mathtt{T}$ such that the sender will have all status report parcels from every node on $\mathcal{P}_{\mathtt{T}}$. Then by Theorem 10.3, the sender can eliminate a corrupt node. At this point, lines (**15.169-177**) essentially call for the protocol to start over, wiping clear all buffers except for the eliminated nodes buffer, which will now contain the identity of a newly eliminated node. The transmission of the latest codeword not yet transmitted then resumes (see comments on (**15.214**)), and the argument can be applied to the new network, consisting of $n-1$ nodes. Since the node-controlling adversary can corrupt at most $n-2$ nodes (the sender and receiver are incorruptible by the conforming assumption), this can happen at most $n-2$ times, yielding the bound of n^2 for the maximum number of failed transmissions. ■

10.1 Main Technical Proof of the Node-Controlling + Edge-Scheduling Protocol

In this section, we aim to prove Lemma 10.3 which states that the sender will be able to eliminate a corrupt node if he has the complete status reports from every node that participated in some failed transmission $\mathtt{T}$. We begin by formally defining the three reasons a transmission may fail, and prove that every failed transmission falls under one of these three cases.

Theorem 10.5. *At the end of any transmission* $\mathtt{T}$*, (at least) one of the following necessarily happens:*

- S1. *The receiver has received at least $D-6n^3$ distinct packets corresponding to the current codeword*
- F2. *S1 does not happen, and the sender has* knowingly[28] *inserted less than D packets*
- F3. *S1 does not happen, the sender has* knowingly[28] *inserted at least D packets, and the receiver has not received any duplicated packets corresponding to the current codeword*
- F4. *S1, F2, and F3 all do not happen*

Proof. That the four cases cover all possibilities (and are disjoint) is immediate. Also, in the case of S1, the receiver can necessarily decode by Lemma 11.20, and hence that case corresponds to a *successful* transmission. Therefore, all failed transmissions must fall under one of the other three cases. ■

Note that case F2 roughly corresponds to *packet duplication*, since the sender is blocked from inserting packets in at least $4D-D$ rounds, indicating jamming that cannot be accounted for by edge failures alone. Case F3 roughly corresponds to *packet deletion*, since the D packets the sender inserted do not reach the receiver (otherwise the receiver could have decoded as by Lemma 11.20), and case F4 corresponds to a mixed adversarial strategy of *packet deletions and duplications.* We treat each case separately in Theorems 10.6, 10.11 and 10.12 below, thus proving Theorem 10.3:

Proof of Theorem 10.3. Theorem 10.5 guarantees that each failed transmission falls under F2, F3, or F4, and the theorem is proven for each case below in Theorems 10.6, 10.11 and 10.12. ■

We declare once-and-for-all that at any time, G will refer to nodes still a part of the network, i.e. nodes that have not been eliminated by the sender.

[28] Recall that by the definition of "inserted" (see 6.6), the sender may not have received confirmation (as in Definition 7.6) that a packet he outputted along some edge was received by the adjacent node. Case F3 requires that the sender has *received confirmation* for at least D packets.

Handling Failures as in F2: Packet Duplication

The goal of this section will be to prove the following theorem.

Theorem 10.6. *Suppose transmission* $\mathtt{T}$ *failed and falls under case F2, and at some later time (after transmission* $\mathtt{T}$ *but before any additional nodes have been eliminated) the sender has received all of the status report parcels from all nodes on* $\mathcal{P}_{\mathtt{T}}$*. Then the sender can eliminate a corrupt node.*

The idea of the proof is as follows. Case F2 of transmission failure roughly corresponds to *packet duplication*: there is a node $N \in G$ who is jamming the network by outputting duplicate packets. Notice that in terms of network potential (see Definition 6.10), the fact that N is outputting more packets than he is inputting means that N will be responsible for illegal increases in network potential. Using the status reports for case F2, which include nodes' signatures on changes of network potential due to packet transfers and re-shuffling, we will catch N by looking for a node who caused a greater increase in potential than is possible if it had been acting honestly. The formal proof of this fact will require some work. We begin with the following definitions:

Definition 10.7. The *conforming* assumption on the node-controlling and edge-scheduling adversaries demand that for every round there is a path connecting the sender and receiver consisting of edges that are "up" and through uncorrupted nodes. We will refer to this path as the active honest path for round $\mathtt{t}$ and denote it by $\mathsf{P}_{\mathtt{t}}$, noting that the path may not be the same for all rounds.

Definition 10.8. We will say that some round $\mathtt{t}$ (of transmission $\mathtt{T}$) is *wasted* if there is an edge $E(A, B)$ on that round's active honest path such that either ***Okay To Send Packet*** (**11.31**) or ***Okay To Receive Packet*** (**11.35**) returned false.

Intuitively, a round is wasted if an edge on the active honest path was prevented from passing a packet either because one of the nodes was blacklisted or because there was important broadcast information that had to be communicated before packets could be transferred.

Lemma 10.9. *There are at most* $4n^3$ *wasted rounds in any transmission* $\mathtt{T}$.

Proof. We will prove this lemma via two claims.

Claim 1. Every wasted round $\mathtt{t}$ falls under (at least) one of the following cases:

1. An edge on $\mathsf{P}_{\mathtt{t}}$ transfers $\Theta_{\mathtt{T}}$ or a parcel of the sender's *Start of Transmission* (SOT) broadcast
2. An edge on $\mathsf{P}_{\mathtt{t}}$ transfers the label of a node to remove from the blacklist
3. An edge on $\mathsf{P}_{\mathtt{t}}$ transfers the information that one of the terminal nodes (on that edge) has the complete status report for a blacklisted node
4. A node on $\mathsf{P}_{\mathtt{t}}$ *learns* a status report parcel for a blacklisted node. More specifically, there is some node $(\widehat{N}, \mathtt{T}', \mathtt{T})$ that was part of the *SOT* broadcast (i.e. the node began the transmission on the sender's blacklist) and some other honest node $N \in G$ such that N learns a new status report parcel from $\widehat{N}$ corresponding to transmission $\mathtt{T}'$.

Proof. Let $\mathtt{t}$ be a wasted round. Denote the active honest path for round $\mathtt{t}$ by $\mathsf{P}_{\mathtt{t}} = N_0 N_1 \dots N_l$. By looking at ***Okay To Send Packet*** and ***Okay To Receive Packet*** (**11.31** and **11.35**), we first argue that that cases 1-3 cover all possible reasons for a wasted round, *except* the possibility that one node is on the other's blacklist. To see this, we go through each line of ***Okay To Send Packet*** and ***Okay To Receive Packet*** and consider what happens along a specified edge on $\mathsf{P}_\mathtt{t}$, noting that by assumption this edge is *active* and the neighboring nodes are *honest*, so the appropriate broadcast parcel will be successfully transferred (**11.15**). In particular, it will be enough to show that for every reason a round may be wasted, there is a node on $\mathsf{P}_\mathtt{t}$ that has broadcast information of type 1-4 (see line (**13.115**)) that it has yet to transfer across an adjacent edge on $\mathsf{P}_\mathtt{t}$, as then we will fall under cases 1-3 of the Claim.

- If there is a node N_i on $\mathsf{P}_\mathtt{t}$ that does not know all parcels of the SOT broadcast (**15.200**), then find the last index $0 \le j < i$ such that N_j knows all of SOT but N_{j+1} does not (j is guaranteed to exist since $S = N_0$ knows all of SOT and N_i does not). Then N_j has broadcast information of type 2 (**13.115**) it has not yet sent along its edge to N_{j+1}.
- If there is a node N_i on $\mathsf{P}_\mathtt{t}$ that knows $\Theta_\mathtt{T}$ or all of SOT but has not yet transferred one of these parcels across an edge of $\mathsf{P}_\mathtt{t}$, or N_i knows the complete status report for some blacklisted node $\widehat{N}$ and N_i has not yet passed this fact along an edge on $\mathsf{P}_\mathtt{t}$, then N_i has broadcast information of type 1, 2, or 4 (**13.115**).
- If there is a node N_i on $\mathsf{P}_\mathtt{t}$ that knows of a node $\widehat{N}$ that should be removed from the blacklist, but it has yet to transfer this information across an edge of $\mathsf{P}_\mathtt{t}$, then N_i has broadcast information of type 3 (**13.115**).

It remains to consider the final reason one of these two functions may return false, namely when there is some N_i on $\mathsf{P}_\mathtt{t}$ that is on the blacklist of either N_{i-1} or N_{i+1}. Let BL_S denote the sender's blacklist at the start of round $\mathtt{t}$.

- If $N_i \notin BL_S$, then there will be some index $0 \le j < i+1$ such that at the start of round $\mathtt{t}$, N_i is not on N_j's blacklist but N_i *is* on N_{j+1}'s blacklist. We may assume that both N_j and N_{j+1} have received the full *start of transmission* broadcast, else we would be in one of the above covered cases. Since N_i is on N_{j+1}'s blacklist, N_i must have begun the transmission on the sender's blacklist (all internal nodes' blacklists are cleared at the end of each transmission (**15.203**) and restored when they receive the SOT broadcast (**15.200**), (**14.137-138**)). However, since N_i is not on N_j's blacklist as of round $\mathtt{t}$ and N_j has received the full SOT broadcast, at some point in $\mathtt{T}$, N_j must have received a parcel from the sender indicating N_i should be removed from the blacklist, as on (**14.147-149**). Since N_j and N_{j+1} are both honest and N_j has received the information that N_i should be removed from the blacklist (but N_{j+1} has *not* received this information yet), it must be that this broadcast information of type 3 (**13.115**) has not yet been successfully passed along $E(N_j, N_{j+1})$ yet. In particular, N_j has broadcast information of priority at least 3 that he has yet to successfully send to N_{j+1}, so he will send a parcel of priority 1, 2, or 3 in round $\mathtt{t}$, which are in turn covered by Statements 1 and 2 of the Lemma.
- If $N_i \in BL_S$, then there exist some $0 \le j < i$ such that N_j does *not* have N_i's complete status report, but N_{j+1} does (since $N_i \in BL_S$ implies S does not have the complete status report, but N_i has its own complete status report in its broadcast buffer, see Statement 2 of Lemma 11.16). Then if N_{j+1} has not yet passed the fact that it has such knowledge along $E(N_{j+1}, N_j)$, then N_{j+1} had broadcast information of type 4, in which case we fall under case 3 of the Claim. On the other hand, if this information has already been passed along $E(N_{j+1}, N_j)$, then Statement 4 of Lemma 11.16 implies that N_j is aware that N_{j+1} knows the complete status report of N_i (who by choice of j is on N_j's blacklist), and hence α will necessarily be set as on (**13.119** or **13.122**) and sent to N_j on

(**13.103**)). Consequently, N_{j+1} will receive α (**13.105**) during Stage 1 communication of round $\mathtt{t}$, and will have broadcast information of type 5 (**13.115**) it has not sent along $E(N_j, N_{j+1})$ yet. This broadcast parcel can then be sent in Stage 2 communication of round $\mathtt{t}$ (**11.15**), and this is covered by case 4 of the Claim. ■

Claim 2. The maximum number of wasted rounds due to Case 1 of Claim 1 is n^3, the maximum number of wasted rounds due to Case 2 of Claim 1 is $n^3/2$, the maximum number of wasted rounds due to Case 3 of Claim 1 is n^3, and the maximum number of wasted rounds due to Case 4 of Claim 1 is n^3.

Proof.

1. $\Theta_{\mathtt{T}}$ is one parcel (**15.179**), and the *SOT* is at most $2n-1$ parcels (**15.200**), so together they are at most $2n$ parcels. Since each honest node will only broadcast each of these parcels at most once across any edge (as long as the broadcast is successful, which it will be if the round is wasted due to Case 1) and there are at most $n^2/2$ such edges, we have that Case 1 can happen at most n^3 times.
2. Lemma 11.23 says that no honest node N will accept more than one distinct parcel (per transmission) that indicates some node $\widehat{N}$ should be removed from the blacklist. Therefore, in terms of broadcasting this information, N will have at most one broadcast parcel per transmission per node $\widehat{N}$ indicating $\widehat{N}$ should be removed from the blacklist. Therefore, it can happen at most n times that an *edge* adjacent to an honest node will need to broadcast a parcel indicating a node to remove. Again since the number of edges is bounded by $n^2/2$, Case 2 can be responsible for a wasted round at most $n^3/2$ times.
3. Lemma 11.24 says that for any node $N \in G$ that has received the full *SOT* broadcast for transmission $\mathtt{T}$, if N is honest then it will transmit along each edge at most once (per transmission) the fact that it knows some $\widehat{N}$'s complete status report. Since each node has at most $n-1$ adjacent edges and there are at most n nodes in G, Case 3 can be responsible for a wasted round at most n^3 times.
4. Notice that Case 4 emphasizes the fact that a node on $\mathsf{P}_{\mathtt{t}}$ *learned* a blacklisted node's status report parcel. Since there are at most $n-1$ blacklisted nodes at any time (see (**15.187-188**) and Claim 11.6), and at most n status report parcels per blacklisted node (see (**14.142-45**) and Lemma 11.7), an honest node can *learn a new* status report parcel at most $n(n-1) < n^2$ times per transmission (see Statement 3 of Lemma 11.16 which says honest nodes will not ever "unlearn" relevant status report parcels). Since there are at most n nodes, Case 4 can be responsible for a wasted round at most n^3 times. ■

Claim 1 guarantees every wasted round falls under Case 1-4, and Claim 2 says these can happen at most $4n^3$ rounds, which proves the lemma. ■

We now define the notation we will use to describe the specific information the status reports contain in the case of F2 (see (**9.12**), (**9.17**), (**9.32**), and (**14.142-145**))[29]:

- $SIG_{A,A}$ denotes the net decrease in A's potential due to re-shuffling packets in the current transmission.

[29] On a technical point, since our protocol calls for internal nodes to keep *old* codeword packets in their buffers from one transmission to the next, packets being transferred during some transmission may correspond to old codewords. We emphasize that the quantities in $SIG_{A,A}$, $SIG[2]$, and $SIG[3]$ include old codeword packets, while $SIG[1]$ and $SIG[p]$ do *not* count old codeword packets (see **11.11** and **12.59-60**).

- $SIG^A[2]_{A,B}$ denotes the net increase in B's potential due to packet transfers across directed edge $E(A,B)$, as signed by B and stored in A's signature buffer ((**12.75**), (**11.11**), and (**11.07**)).
- $SIG^A[2]_{B,A}$ denotes the net decrease in B's potential due to packet transfers across directed edge $E(B,A)$, as signed by B and stored in A's signature buffer. Notice that $SIG^A[2]_{B,A}$ is measured as a *positive quantity*, see lines (**12.60**), (**12.62**), and (**12.74**).
- $SIG^A[3]_{A,B}$ denotes the net decrease in A's potential due to packet transfers across directed edge $E(A,B)$, which is signed by A and stored its own signature buffer. Notice that $SIG^A[3]_{A,B}$ is measured as a *positive quantity*, see line (**12.49**).
- $SIG^A[3]_{B,A}$ denotes the net increase in A's potential due to packet transfers across directed edge $E(B,A)$, which is signed by A and stored its own signature buffer (**12.75**).

Lemma 10.10. *Suppose transmission* $\mathtt{T}$ *failed and falls under case F2, and at some later time (after transmission* $\mathtt{T}$ *but before any additional nodes have been eliminated) the sender has received all of the status reports from every node on* $\mathcal{P}_{\mathtt{T}}$*. Then one of the following two things happens:*

1. *There is some node $A \in G$ whose status report indicates that A is corrupt*[30]*.*
2. *There is some $A \in G$ whose potential at the start of* $\mathtt{T}$ *plus the net increase in potential during* $\mathtt{T}$ *is smaller than its net decrease in potential during* $\mathtt{T}$*. More specifically, note that A's net* ***increase*** *in potential, as claimed by itself, is given by:*

$$\sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{B,A}$$

Also, A's net ***decrease*** *in potential, as documented by all of its neighbors and its own loss due to re-shuffling, is given by:*

$$SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B}$$

Then case 2) says there exists some $A \in G$ such that:

$$4n^3 - 4n^2 + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{B,A} \quad < \quad SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B}, \tag{16}$$

where the $4n^3 - 4n^2$ term on the LHS is an upper bound for the maximum potential a node should have at the outset of a transmission (see proof of Claim 6.2).

Proof. The idea of the proof is to use Lemma 11.12, which argues that in the absence of malicious activity, the network potential should drop by at least n every (non-wasted) round in which the sender is unable to insert a packet. Then since the sender could not insert a packet in at least $3D$

[30] This includes, but is not limited to: 1) The node has returned a (value, signature) pair, where the value is not in an appropriate domain; 2) The node has returned non-zero values indicating interaction with blacklisted or eliminated nodes; 3)The node has reported values for $SIG^A[3]_{S,A}$ that are inconsistent with the sender's quantity $SIG^S[2]_{S,A}$; or 4) The node has returned outdated information in their status report. By "outdated" information, we mean that as part of its status report, A returned a (value, signature) pair using a signature he received in round $\mathtt{t}$ from one of A's neighbors N, but in N's status report, N provided a (value, signature) pair from A indicating they communicated *after* round $\mathtt{t}$ and that A was necessarily using an outdated signature from N.

rounds (case F2 states the sender inserted fewer than D packets in the $4D$ rounds of the transmission) and since there are at most $4n^3$ wasted rounds per transmission, the network potential should have dropped by at least $(n)(3D - 4n^3) > 2nD + 8n^4$ (since $D = \frac{6n^3}{\lambda} > 12n^3$ as $\lambda < 1/2$). However, this is impossible, since the maximum network potential in the network at the start of the transmission (which is upper bounded from the capacity of the network) is $4n^4$ (Lemma 6.2) plus the maximum amount of network potential increase during transmission $\mathtt{T}$ is $2nD$ (since the sender inserted fewer than D packets at maximum height $2n$), and hence the sum of these is less than $2nD+8n^4$, resulting in a negative network potential. Since network potential can never be negative, there must be illegal increases to network potential not accounted for above, and the node responsible for these increases is necessarily corrupt. We now formalize this argument, showing how to find such an offending node and prove it is corrupt.

Let β denote the number of rounds in transmission $\mathtt{T}$ that the sender was blocked from inserting any packets, and $\mathcal{P}$ denote the participating list for $\mathtt{T}$.

Obs. 1: If there exists $A \in \mathcal{P}$ such that one of the following inequalities is not true, then A is corrupt.

$$0 \leq SIG_{A,A} \qquad 0 \leq \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^A[2]_{B,A} - SIG^A[3]_{B,A})$$

Proof. The above inequalities state that for honest nodes, the potential changes due to reshuffling and packet transfers are strictly non-positive (this was the content of Lemma 6.11). This observation is proved rigorously as Statements 4 and 5 of Lemma 11.9 in Section 11. □

Obs. 2: The increase in network potential due to packet insertions is at most $2nD + 2n^2$. More precisely, either there exists a node $A \in G$ such that the sender can eliminate A, or the following inequality is true:

$$\sum_{A \in \mathcal{P} \setminus S} SIG^A[3]_{S,A} < 2nD + 2n^2 \tag{17}$$

Proof. By hypothesis, the sender *knowingly* inserted less than D packets in transmission $\mathtt{T}$, and each packet can increase network potential by at most $2n$. The sum on the LHS of (17) represents the increase in potential claimed by nodes participating in $\mathtt{T}$ caused by packet insertions. This quantity should match the sender's perspective of the potential increase (which is at most $2nD$), with the exception of potential increases caused by packets that were inserted but S did not received confirmation of receipt (see Definition 7.6). There can be at most one such packet per edge, causing an additional potential increase of at most $2n$ per edge. Adding this additional potential increase to the maximum increase of $2nD$ of the sender's perspective is the RHS of (17). The formal proof can be found in Lemma 11.15 in Section 11. □

Obs. 3: $\beta \geq 3D - n$. (Recall that β denotes the number of blocked rounds in $\mathtt{T}$.)

Proof. Since the sender knowingly inserted fewer than D packets, there could be at most n packets (one packet per edge) that was inserted unbeknownst to S, and hence the sender must have been blocked for (at least) all but $D + n$ of the rounds of the transmission. Since the number of rounds in a transmission is $4D$ (**11.02**), we have that $\beta \geq 3D - n$. □

Let $\mathcal{H}_{\mathtt{T}} \subseteq \mathcal{P}_{\mathtt{T}}$ denote the subset of participating nodes that are honest (the sender is of course oblivious as to which nodes are honest, but we will nevertheless make use of $\mathcal{H}_{\mathtt{T}}$ in the following argument). For notational convenience, since transmission $\mathtt{T}$ is fixed, we suppress the subscript and write simply $\mathcal{H}$ and $\mathcal{P}$. We make the following simple observations:

Obs. 4: The following inequality is true:

$$2nD + 4n^4 - 4n^3 + 2n^2 < \sum_{A \in \mathcal{H} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{H} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^A[2]_{B,A} - SIG^A[3]_{B,A}) \tag{18}$$

Proof. This follows immediately from Observation 3 and Lemma 11.12, since:

$$\begin{aligned} n(\beta_{\mathtt{T}} - 4n^3) &\geq n(3D - n - 4n^3) \\ &\geq 2nD + 4n^4 - 4n^3 + 2n^2, \end{aligned}$$

where the first inequality is Observation 3, and the second follows because $D = 6n^3/\lambda \geq 8n^3 \geq 8n^3 - 4n^2 + 3n$. □

Obs. 5: Either a corrupt node can be identified as in Obs. 1 or 2, or there is some $A \in \mathcal{P}$ such that:

$$4n^3 - 4n^2 < SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B} - SIG^A[3]_{B,A} \tag{19}$$

Proof. Consider the following inequalities:

$$\begin{aligned} 2nD + 4n^4 - 4n^3 + 2n^2 &< \sum_{A \in \mathcal{H} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{H} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^A[2]_{B,A} - SIG^A[3]_{B,A}) \\ &\leq \sum_{A \in \mathcal{P} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{P} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^A[2]_{B,A} - SIG^A[3]_{B,A}) \\ &= \sum_{A \in \mathcal{P} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{P} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^B[2]_{A,B} - SIG^A[3]_{B,A}) \end{aligned} \tag{20}$$

Above, the top inequality follows from Obs. 4, the second inequality follows from Obs. 1, and the third line is a re-arranging and re-labelling of terms. Subtracting $2nD + 2n^2$ from both sides:

$$\begin{aligned} 4n^4 - 4n^3 &< \sum_{A \in \mathcal{P} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{P} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^B[2]_{A,B} - SIG^A[3]_{B,A}) \; - 2nD - 2n^2 \\ &< \sum_{A \in \mathcal{P} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{P} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^B[2]_{A,B} - SIG^A[3]_{B,A}) + \sum_{A \in \mathcal{P} \setminus S} -SIG^A[3]_{S,A} \\ &= \sum_{A \in \mathcal{P} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{P} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^B[2]_{A,B} - SIG^A[3]_{B,A}) + \\ &\qquad \sum_{A \in \mathcal{P} \setminus S} (SIG^S[2]_{A,S} - SIG^A[3]_{S,A}) \\ &= \sum_{A \in \mathcal{P} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{P} \setminus S} \sum_{B \in \mathcal{P} \setminus A} (SIG^B[2]_{A,B} - SIG^A[3]_{B,A}) \end{aligned} \tag{21}$$

Above, the top inequality is from (20), the second follows from Obs. 2, the third line is because $SIG^S[2]_{A,S} = 0$ for all $A \in G$ (S never *receives* a packet from anyone, see (**11.21-22**)), and the final line comes from combining sums. Using an averaging argument, this implies there is some $A \in \mathcal{P} \setminus S$ such that:

$$4n^3 - 4n^2 < SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} (SIG^B[2]_{A,B} - SIG^A[3]_{B,A}), \tag{22}$$

which is (19). □

Therefore, if a node cannot be eliminated as in Obs. 1 or 2 (which are covered by Case 1 of Lemma 10.10), then Obs. 5 implies that Case 2 of Lemma 10.10 is true. ■

Proof of Theorem 10.6. This Theorem now follows immediately from Lemma 10.10 and the fact that a node $A \in G$ for which (16) is true is necessarily corrupt. Intuitively, such a node $A \in G$ is corrupt since the potential decrease at A is higher than can be accounted for by A's potential at the outset of $\mathtt{T}$ plus the potential increase due to packet insertions from the sender. The formal statement and proof of this fact is the content of Corollary 11.14. ■

Handling Failures as in F3: Packet Deletion

The goal of this section will be to prove the following theorem.

Theorem 10.11. *Suppose transmission* $\mathtt{T}$ *failed and falls under case F3, and at some later time (after transmission* $\mathtt{T}$ *but before any additional nodes have been eliminated) the sender has received all of the status report parcels from all nodes on* $\mathcal{P}_{\mathtt{T}}$. *Then the sender can eliminate a corrupt node.*

The idea of the proof is as follows. Case F3 of transmission failure roughly corresponds to *packet deletion*: there is a node $N \in G$ who is deleting some packets transferred to it instead of forwarding them on. Using the status reports for case F3, which include nodes' signatures on the net number of packets that have passed across each of their edges, we will catch N by looking for a node who input more packets than it output, and this difference is greater than the buffer capacity of the node.

Proof. We first define the notation we will use to describe the specific information the status reports contain in the case of F3 ((**9.17**), (**9.32**), and (**14.144**)):

- $SIG^A[1]_{A,B}$ denotes the net number of packets that have travelled across directed edge $E(A,B)$, as signed by B and stored in A's (outgoing) signature buffer.
- $SIG^A[1]_{B,A}$ denotes the net number of packets that have travelled across directed edge $E(B,A)$, as signed by B and stored in A's (incoming) signature buffer.

By the third Statement of Lemma 11.13, either a corrupt node can be eliminated, or the following is true for all $A, B \in G$:

$$|SIG^A[1]_{B,A} - SIG^B[1]_{B,A}| \leq 1 \qquad \text{and} \qquad |SIG^A[1]_{A,B} - SIG^B[1]_{A,B}| \leq 1$$

Then summing over all $A, B \in \mathcal{P}$:

$$\sum_{A,B \in \mathcal{P}, A \neq B} |SIG^A[1]_{B,A} - SIG^B[1]_{B,A}| \ \leq \ n^2 \tag{23}$$

This in turn implies that:

$$
\begin{aligned}
-n^2 &\leq \sum_{A,B\in\mathcal{P},A\neq B} (SIG^A[1]_{B,A} - SIG^B[1]_{B,A}) \\
&= \sum_{A\in\mathcal{P}} \sum_{B\in\mathcal{P}\setminus A} (SIG^A[1]_{B,A} - SIG^A[1]_{A,B}) \\
&= \sum_{B\in\mathcal{P}\setminus R} SIG^R[1]_{B,R} - \sum_{B\in\mathcal{P}\setminus S} SIG^S[1]_{S,B} + \sum_{A\in\mathcal{P}\setminus\{R,S\}} \sum_{B\in\mathcal{P}\setminus A} \big(SIG^A[1]_{B,A} - SIG^A[1]_{A,B}\big) \\
&\leq -6n^3 + \sum_{A\in\mathcal{P}\setminus\{R,S\}} \sum_{B\in\mathcal{P}\setminus A} (SIG^A[1]_{B,A} - SIG^A[1]_{A,B})
\end{aligned}
$$

The first inequality is from (23), the second line is from re-labelling and re-arranging terms, the third line comes from separating out the terms $A = S$ and $A = R$ and noting that $SIG^R[1]_{R,B} = SIG^S[1]_{B,S} = 0$ (since the receiver will never output packets to other nodes and the sender will never input packets, see (**11.16-20**) and (**11.21-22**)), and the final inequality is due to the fact that we are in case F3, so the sender knowingly inserted D packets, but the receiver received fewer than $D - 6n^3$ packets corresponding to the current codeword[31]. Using an averaging argument, we can find some $A \in G$ such that:

$$
4n^2 - 8n < 6n^2 - n < \sum_{B\in\mathcal{P}\setminus A} (SIG^A[1]_{B,A} - SIG^A[1]_{A,B}), \tag{24}
$$

where the first inequality is obvious. Statement 7 of Lemma 11.9 now guarantees that A is corrupt[32].

■

Handling Failures as in F4: Packet Duplication + Deletion

The goal of this section will be to prove the following theorem.

Theorem 10.12. *Suppose transmission* T *failed and falls under case F4, and at some later time (after transmission* T *but before any additional nodes have been eliminated) the sender has received all of the status report parcels from all nodes on* $\mathcal{P}_{\mathtt{T}}$*. Then the sender can eliminate a corrupt node.*

The idea of the proof is as follows. Case F4 of transmission failure roughly corresponds to packet duplication *and* packet deletion: there is a node $N \in G$ who is replacing valid packets with copies of old packets it has already passed on. Therefore, simply tracking potential changes and net packets into and out of N will not help us to locate N, as both of these quantities will be consistent with honest behavior. Instead, we use the fact that case F4 implies that the receiver will have received some packet p (from the current codeword) twice. We will then use the status reports for case F4, which include nodes' signatures on the net number of times p has crossed each of their edges, to find a corrupt node N by looking for a node who output p more times than it input p.

[31] More precisely, F3 states that the sender knowingly inserted at least D packets and the receiver did not receive any packet (from the current codeword) more than once. By Fact 1', since we are not in case S1, the receiver got fewer than $D - 6n^3$ distinct packets corresponding to the current codeword.

[32] Intuitively, A must be corrupt since the sum on the RHS of (24) represents the net number of packets A input minus the number of packets A output. Since this difference is larger than the capacity of A's internal buffers, A must have deleted at least one packet and is necessarily corrupt.

Proof. By definition of F4, the receiver received some packet p (corresponding to the current codeword) at least twice. Therefore, when (**15.178-179**) is reached, the receiver will send the label of p back to the sender (which reaches S by the end of the transmission by Lemma 11.19), and this is in turn broadcasted as part of the sender's *start of transmission* broadcast in the following transmission ((**15.190-192**) and (**15.200**)). We will use the following notation to describe the specific information the status reports contain in the case of F4 (see (**9.17**) and (**9.32**)):

- $SIG^A[p]_{A,B}$ denotes the net number of times p has travelled across directed edge $E(A,B)$, as signed by B and stored in A's (outgoing) signature buffer.
- $SIG^A[p]_{B,A}$ denotes the net number of times p has travelled across directed edge $E(B,A)$, as signed by B and stored in A's (incoming) signature buffer.

Consider the following string of equalities:

$$\begin{aligned}
0 &= \sum_{A\in\mathcal{P}_{\mathrm{T}}}\sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^A[p]_{B,A} - SIG^A[p]_{B,A}) \\
&= \sum_{A\in\mathcal{P}_{\mathrm{T}}}\sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}) \\
&= \sum_{A\in\mathcal{P}_{\mathrm{T}}\setminus\{R,S\}}\sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}) + \sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{R,B} - SIG^R[p]_{B,R}) + \\
&\qquad \sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{S,B} - SIG^S[p]_{B,S}) \\
&= \sum_{A\in\mathcal{P}_{\mathrm{T}}\setminus\{R,S\}}\sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}) + \sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{S,B} - SIG^R[p]_{B,R}) \qquad (25)
\end{aligned}$$

The first equality is trivial, the second equality comes from re-labelling and rearranging the terms of the sum, the third comes from separating out the $A=S$ and $A=R$ terms, and the final equality results from the fact that R never outputs packets and S never inputs packets, and hence they will never sign non-zero values for $SIG[p]_{R,B}$ or $SIG[p]_{B,S}$, respectively (see (**11.16-20**) and (**11.21-22**)). Because p was received by R at least twice (by choice of p) and S will never send any packet to more than one node[33], we have that:

$$\sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{S,B} - SIG^R[p]_{B,R}) \le -1 \qquad (26)$$

Plugging this into (25) and rearranging:

$$1 \le \sum_{A\in\mathcal{P}_{\mathrm{T}}\setminus\{R,S\}}\sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}) \qquad (27)$$

By an averaging argument, there must be some $A\in\mathcal{P}_{\mathrm{T}}\setminus\{R,S\}$ such that:

$$1 \le \sum_{B\in\mathcal{P}_{\mathrm{T}}}(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}) \qquad (28)$$

Now Statement 8 of Lemma 11.9 says that A is necessarily corrupt[34]. ∎

[33] This was proven in Observations 2-3 of Lemma 7.13 for the edge-scheduling protocol. However, the proofs of these observations remain valid in the (node-controlling+edge-scheduling) model because the sender is honest (by the conforming adversary assumption).

[34] Intuitively, A is corrupt since (28) says that it has output p more times than it input p.

11 Node-Controlling + Edge-Scheduling Protocol: Pseudo-Code Intensive Proofs

In this section, we give detailed proofs that walk through the pseudo-code of Figures 9 - 15 to argue very basic properties the protocol satisfies. The following lemma will relieve the need to re-prove many of the lemmas of Sections 6 and 7.

Lemma 11.1. *Differences between the edge-scheduling adversary protocol and the (node-controlling + edge-scheduling) adversary protocol all fall under one of the following cases:*

1. *Extra variables in the Setup Phase*
2. *Length of transmission and codeword being transmitted for the current transmission*
3. *Need to authenticate signatures on packets, as on (**11.08**) and (**12.68**)*
4. *Need to check if it is okay to send/receive packets, as on (**12.59**) and (**12.63**)*
5. *Broadcasting information, i.e. transmission of broadcast parcels and modifications of Broadcast Buffer, Data Buffer, and Signature Buffer*

Furthermore, differences as in Cases 3 and 4 are identical to having an edge-fail in the edge-scheduling adversary protocol. Also, differences as in Case 5 affect the routing protocol only insofar as their affect on Cases 3 and 4 above. Furthermore, between any two honest nodes, the authentications of Case 3 never fail, and Case 4 failures correspond to "wasted" rounds (see Definition 10.8).

Proof. Comparing the pseudo-code of Figures **5**, **6**, and **7** to Figures **11**, **12**, and **15**, as emphasized by line numbers in **bold face** in the latter three, it is clear that all differences fall under Cases 1-5 of the lemma. Also, all of the other methods in Figures **13-15** fall under Cases 4 and 5.

As for the differences as in Cases 3 and 4, it is clear that failing ***Verify Signature One*** on (**12.86-87**) is equivalent to the edge failing during Stage 2 (i.e. as if $p = \perp$ on (**12.62**) causing (**12.69**) to fail); failing ***Verify Signature Two*** on (**12.89-90**) is equivalent to the edge failing during Stage 1 (since this sets H_{IN} and RR to $\perp$ on (**12.90**), which is equivalent to the communication on (**5.06**) not being received); failing ***Okay to Send Packet*** on (**12.59**) is equivalent to the edge failing during Stage 2 (so that nothing is received on lines (**11.22**/**12.62**)); and failing ***Okay to Receive Packet*** on (**12.63**) is equivalent to the edge failing during Stage 1 (i.e. as if nothing is received on (**11.13**), so that $H_{OUT} = \perp$ on (**12.63**)). Finally, differences as in Case 5 do not directly affect routing (except their affects captured by Cases 3 and 4) since the transfer of broadcast parcels and maintenance of the related buffers (signature, broadcast, and data buffers) happen independently of the routing of codeword packets. This is evident by investigating the relevant **bold** lines in Figures **11**, **12**, and **15**.

The second part of the last sentence is true by definition of wasted (see Definition 10.8), and the first part follows from lines (**11.11**), (**12.49**), (**12.60**), (**12.75**), and Lemma 11.17. ■

Lemma 11.2. *The domains of all of the variables in Figures 9 and 10 are appropriate. In other words, the protocol never calls for more information to be stored in an honest node's variable (buffer, packet, etc.) than the variable has room for.*

Proof. The proof for variables and buffers that also appear in the edge-scheduling protocol follows from Lemmas 7.1 and 7.2, since all differences between the edge-scheduling protocol and the (node-controlling + edge-scheduling) protocol are equivalent to an edge-failure (Lemma 11.1). So it remains to prove the lemma for the new variables appearing in Figures 9 and 10 (i.e. the **bold** line numbers). The distribution of public and private keys (**9.14**) is performed by a trusted third party, so these variables are as specified. Below, when we refer to a specific node's variable, we implicitly assume the node is honest, as the lemma is only concerned about honest nodes.

Bandwidth P (**9.06**). We look at all transfers along each directed edge in each stage of any round. In Stage 1, this includes the transfer of H_{OUT}, H_{FP}, FR (**11.06**), c_{bp}, α (**11.04**), and the seven signed items on (**11.11**). All of these have collective size $O(k + \log n)$ ((**9.03-04**), (**9.21**), (**9.23**), (**9.24**), (**9.27**), (**9.29**), (**9.32**), (**9.35**), and (**9.36**)). In Stage 2, this includes the transfer of the seven items on (**12.60**) and bp (**11.15**). Collectively, these have size $O(k + \log n)$ ((**9.03-04**), (**9.17**), (**9.18**), (**9.21**), and (**9.42**)).

Potential Lost Due to Re-Shuffling $SIG_{N,N}$ (**9.12**). This is initialized to zero on (**10.46**), after which it is only updated on (**7.76**), (**12.50**), (**12.80**), (**12.82**), (**14.128**), (**14.133**), (**14.141**), and (**14.146**). The first four of these increment $SIG_{N,N}$ by at most $2n$, and the latter four all reset $SIG_{N,N}$ to zero. We will see in Lemma 11.16 below that $SIG_{N,N}$ will always represent the potential lost due to re-shuffling in at most one failed transmission, and consequently $SIG_{N,N}$ is polynomial in n, as required.

Broadcast Parcel bp to Receive (**9.27**). This is initialized to $\perp$ on (**10.58**), after which it is only updated on (**13.93**). Either no value was received on (**13.93**) (in which case $bp = \perp$), or it corresponds to the value sent on (**13.97**). As discussed below, the value of bp sent on (**13.97**) lies in the appropriate domain, and hence so does bp.

Broadcast Buffer Request α (**9.28**). This is initialized to $\perp$ on (**10.58**), after which it is only updated as in ***Broadcast Parcel to Request*** (**13.117-122**). On (**13.117**), α is set to $\perp$, and on (**13.119**) and (**13.122**), α includes the label of a node and a status report parcel (see **14.142-145**), and so α is bounded by $O(k + \log n) = P$ as required.

Outgoing Verification of Broadcast Parcel Bit c_{bp} (**9.29**). This is initialized to zero on (**10.58**), after which it is only updated as on (**12.40**) and (**13.111**), where it clearly lies in the appropriate domain.

Broadcast Parcel bp to Send (**9.42**). This is initialized to $\perp$ on (**10.52**), after which it is only updated as in ***Determine Broadcast Parcel to Send*** (**13.115**). Looking at the six types of broadcast parcels on line (**13.115**) and comparing the corresponding domains of these variables in Figures **9** and **10**, we see that in each case, bp can be expressed in $O(k+\log n) = P$ bits.

Incoming Verification of Broadcast Parcel Bit c_{bp} (**9.43**). This is initialized to zero on (**10.52**), after which it is only updated as on (**13.105**) and (**13.108**). The value it takes on (**13.105**) will either be set to zero (if no value was received), or it will equal the value of c_{bp} sent on (**13.101**), which as shown above is either a one or zero. Meanwhile, the value it takes on (**13.108**) is zero, so at all times c_{bp} equals one or zero, as required.

First Parcel of Start of Transmission Broadcast $\Omega_{\mathtt{T}}$ (**10.66**). This is initialized to $(0,0,0,0)$ on (**10.72**) and is only changed on (**15.174**), (**15.184**), (**15.192**), (**15.195**), and (**15.198**). In all of these cases, it is clear that $\Omega_{\mathtt{T}}$ can be expressed in $O(\log n)$ bits, as required.

Number of Rounds Blocked $\beta_{\mathtt{T}}$ (**10.67**). This is initialized to zero on (**10.71**) and is only changed on (**11.27**), (**15.172**), and (**15.201**). Notice that in the latter two cases, $\beta_{\mathtt{T}}$ is reset to zero, while $\beta_{\mathtt{T}}$ can only be incremented by one on (**11.27**) at most $4D$ times per transmission by (**11.02**). Since either line (**15.172**) or line (**15.201**) is reached at the end of every transmission (in the case a node is not eliminated as on line (**14.163**) or (**14.168**), line (**15.201**) will be reached by the call on (**11.29**)), $\beta_{\mathtt{T}} \in [0..4D]$ at all times, as required.

Number of Failed Transmissions F (**10.68**). This is initialized to zero on (**10.71**) and is only changed on (**15.172**) and (**15.186**). Notice that F is only incremented by one as on line (**15.186**) when a transmission fails. As was shown in Theorem 10.4, there can be at most $n-1$ failed transmissions before a node can necessarily be eliminated, in which case F is reset to zero on (**15.172**).

Participating List $\mathcal{P}_{\mathtt{T}}$ (**10.69**). This is initialized to G on (**10.73**) and is only changed on (**15.173**) and (**15.187**), where it is clear each time that $\mathcal{P}_{\mathtt{T}} \subseteq G$ in both places.

End of Transmission Parcel $\Theta_{\mathtt{T}}$ (**10.77**). This is initialized to $\perp$ on (**10.79**) and is only changed on (**15.179**), where it is clear that $\Omega_{\mathtt{T}}$ can be expressed in $O(k+\log n)$ bits as required (packets have size $O(k+\log n)$, and the index of a transmission requires $O(\log n)$ bits).

Broadcast Buffer BB (**9.08**). We treat the sender's broadcast buffer separately below, and consider now only the broadcast buffer of any internal node or the receiver. Notice that the broadcast buffer is initially empty (**10.46**). Looking at all places information is *added* to BB (lines (**13.106-107**), (**14.125**), (**14.127**), (**14.130**), (**14.136**), (**14.138**), (**14.142-145**), (**14.148-149**), (**14.154**), and (**14.155**)), we see that there are 7 kinds of parcels stored in the broadcast buffer, as listed on (**13.115**) (the 7^{th} type is to indicate which parcel to send across each edge, as on (**13.106**)). We look at each one separately, stating the maximum number of bits it requires in any broadcast buffer. For all of the items below, the comments on (**14.123**) ensure that there are never duplicates of the same parcel in BB at the same time, and also that every parcel in BB has associated with it $n-1$ bits to indicate which edges the parcel has travelled across (see e.g. (**13.107**), (**14.125**), (**14.127**), (**14.130**), (**14.136**), (**14.138**), (**14.148**), and (**14.154**)). Totaling all numbers below, we see that the BB needs to hold at most n^2+5n broadcast parcels, with each parcel needing to record which of the $n-1$ edges it has traversed, which proves the domain on (**9.08**) is correct.

1. Receiver's End of Transmission Parcel $\Theta_{\mathtt{T}}$. This is added to a node's broadcast buffer on (**14.125**), and removed on (**15.203**). Since every internal node and the receiver will reach (**15.203**) at the end of every transmission ((**11.30**) and (**15.203**)), and by the inforgibility of the signature scheme, there is only one valid $\Theta_{\mathtt{T}}$ per transmission T. Therefore, each node will have at most one broadcast parcel of this type in BB at any time.
2. Sender's Start of Transmission Parcels. These are added to a node's broadcast buffer on (**14.127**), (**14.130**), (**14.136**), and (**14.138**), and they are removed on

(**15.203**). Since every internal node and the receiver will reach (**15.203**) at the end of every transmission ((**11.30**) and (**15.203**)), by the inforgibility of the signature scheme, for every transmission $\mathtt{T}$, there is only one valid $\Omega_{\mathtt{T}}$ in BB at any time. Notice that $\Omega_{\mathtt{T}}$ can hold up to 1 parcel for 200a, $n-1$ valid parcels for 200b and 200d together, and up to $n-1$ valid parcels for 200c (see (**15.200**), and use the fact that $S \notin EN, BL$ and Theorem 10.4). Therefore, each node will have at most $2n$ broadcast parcels of this type in BB at any time.

3. Label of a Node to Remove from the Blacklist. Parcels of this nature are added to a node's broadcast buffer on (**14.148**) and removed on (**15.203**). Since every internal node and the receiver will reach (**15.203**) at the end of every transmission ((**11.30**) and (**15.203**)), we argue that in any transmission, every node will have at most $n-1$ parcels in their broadcast buffer corresponding to the label of a node to remove from the blacklist. To see this, we argue that the sender will add $(\widehat{N}, 0, \mathtt{T})$ to his broadcast buffer as on (**14.165**) at most once for each node $\widehat{N} \in \mathcal{P} \setminus S$ per transmission, and then use the inforgibility of the signature scheme to argue each node will add a corresponding broadcast parcel to their broadcast buffer as on (**14.148**) at most $n-1$ times. That the sender will enter line (**14.165**) at most once per node per transmission is clear since once the sender has reached (**14.165**) for some node $\widehat{N}$, the node will be removed from his blacklist on (**14.166**), and nodes are not re-added to the blacklist until the end of any transmission, as on (**15.188**). Therefore, once the sender has received some node $\widehat{N}$'s complete status report as on (**14.164**), that same line cannot be entered again by the same node $\widehat{N}$ in the same transmission. In summary, there are at most $n-1$ broadcast parcels of this type in any node's broadcast buffer at any time.
4. The Label of a Node $\widehat{N}$ Whose Status Report is Known to N. We show that for any node $N \in \mathcal{P} \setminus S$, there are at most $(n-1)$ broadcast parcels of type 4 (**13.115**) in BB at any time[35]. This follows from the same argument as above, where it was shown that (**14.164**) can be true at most once per node per transmission. The inforgibility of the signature scheme ensures that the same will be true for internal nodes regarding line (**14.155**), and since this is the only line on which broadcast parcels of this kind are added to BB, this can happen at most $n-1$ times per transmission. However, we are not yet done with this case, because broadcast information of this type is *not* removed from BB at the end of each transmission like the above forms of broadcast information. Therefore, we fix $\widehat{N} \in G$, and show that if N adds a broadcast parcel to BB of form $(N, \widehat{N}, \mathtt{T}')$ as on (**14.155**) of transmission $\mathtt{T}$, then necessarily BB was *not* already storing a broadcast parcel of form $(N, \widehat{N}, \mathtt{T}'')$ for some other $\mathtt{T}'' \neq \mathtt{T}'$ (if $\mathtt{T}'' = \mathtt{T}'$, then there is nothing to show, as nothing new will be added to BB by the comments on **14.123**).

   For the sake of contradiction, suppose that BB is already storing a parcel of form $(N, \widehat{N}, \mathtt{T}'')$ when (**14.155**) of transmission $\mathtt{T}$ is entered and N is called to add $(N, \widehat{N}, \mathtt{T}')$ to BB for some $\mathtt{T}' \neq \mathtt{T}''$. Since (**14.155**) is reached, we must have that (**14.152**) was satisfied for the *bp* appearing there. In particular, $\widehat{N}$ is on N's version of the blacklist. Since the blacklist is cleared at the end of every transmission (**15.203**), it must be that $(\widehat{N}, \mathtt{T}', \mathtt{T})$ was added to N's version of the blacklist during the *SOT* broadcast for the current transmission $\mathtt{T}$, as on (**14.137-138**). Therefore, all parcels in BB of form

[35]The $(n-1)$ comes from the fact that there are no status reports for the sender.

$(N, \widehat{N}, \mathtt{T}'')$ for $\mathtt{T}'' \neq \mathtt{T}'$ should have been removed from BB on line (**14.139**), yielding the desired contradiction.

5-6. STATUS REPORT PARCELS. We fix $N \in G$ and show that for every $\widehat{N} \in \mathcal{P} \setminus \{S, N\}$, there are at most n status report parcels corresponding to $\widehat{N}$ in N's broadcast buffer, and hence N's broadcast buffer will hold at most $n(n-1)$ status report parcels at any time. Since a single node's status report for a single transmission consists of at most n parcels (see lines (**14.142-145**)[36]), it will be enough to show that for every $\widehat{N} \in \mathcal{P} \setminus S$, at all times N's broadcast buffer only holds status report parcels for $\widehat{N}$ corresponding to a *single* failed transmission $\mathtt{T}'$.

For the sake of contradiction, suppose that during some transmission $\mathtt{T}$, there is some node $\widehat{N} \in \mathcal{P} \setminus S$ and two transmissions $\mathtt{T}'$ and $\mathtt{T}''$ such that N's broadcast buffer holds at least one status report parcel for $\widehat{N}$ from both $\mathtt{T}'$ and $\mathtt{T}''$. Notice that status report parcels are only added to BB as on (**14.154**), and without loss of generality suppose that the status report parcel of $\widehat{N}$ corresponding to $\mathtt{T}''$ was already in BB when one corresponding to $\mathtt{T}'$ is added to BB as on (**14.154**) of transmission $\mathtt{T}$. As was argued above, since (**14.154**) is reached in $\mathtt{T}$, (**14.152**) must have been satisfied, and since N's blacklist is cleared at the end of every transmission (**15.203**), it must be that a broadcast parcel of form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T})$, adding $\widehat{N}$ to N's version of the blacklist, was received earlier in transmission $\mathtt{T}$. Notice that necessarily $\widehat{\mathtt{T}} = \mathtt{T}'$, since otherwise line (**14.153**) will not be satisfied. But then since $\mathtt{T}'' \neq \mathtt{T}'$, all status report parcels of $\widehat{N}$ corresponding to transmission $\mathtt{T}''$ should have been removed from BB on (**14.139**), yielding the desired contradiction.

Now for a N's *own* status report parcels, these are added to BB on (**14.142-145**). Investigating lines (**14.137**), (**14.139**), and (**14.140**), we see that status reports of N can occupy BB for at most one failed transmission.

7. REQUESTED PARCEL FOR EACH EDGE. For any edge $E(A, N)$, N will have at most one copy of a parcel like α as on (**13.106**) at any time, since the old version of α is simultaneously deleted when the new one is added on (**13.106**). Since each node has $(n-1)$ incoming edges, BB need hold at most $n-1$ parcels of this form at any time.

Data Buffer DB, Eliminated List EN, and Blacklist BL (**9.09-11**). We treat the sender's broadcast buffer separately below, and consider now only the data buffer of any internal node or the receiver. The data buffer (which includes the blacklist and list of eliminated nodes) is initially empty (**10.46**). A node N's data buffer holds three different kinds of information: blacklist, list of eliminated nodes, and for each neighbor $\widehat{B} \in G$, a list of nodes $\widehat{N} \in G$ for which $\widehat{B}$ knows the complete status report (see item 4 on line (**13.115**)). Below, we show that these contribute at most $n-1$, $n-1$, and $(n-1)^2$ parcels (respectively), so that DB requires at most n^2 parcels at any time.

BLACKLIST BL. Each entry of BL is initialized to $\perp$ on (**10.46**), and BL is only modified on lines (**14.134**), (**14.138**), (**14.148**), and (**15.203**). BL is an array with $n-1$ entries, indexed by the nodes in $\mathcal{P} \setminus S$. When a node $(\widehat{N}, \mathtt{T})$ is added to BL as on (**14.138**), this means that the entry of BL corresponding to $\widehat{N}$ is switched to be $\mathtt{T}$. When a node $(\widehat{N}, \mathtt{T})$

[36]We assume that the signature buffer information for two directed edges $E(A, B)$ and $E(B, A)$ are combined into one status report parcel.

is removed from BL as on (**14.148**), this means that the entry of BL corresponding to $\widehat{N}$ is switched to $\perp$. Finally, when BL is to be cleared as on (**14.134**) and (**15.203**), this means that BL each entry of BL is set to $\perp$. Thus, in all cases, $BL \in [1..n-1] \times \{0,l\}^P$ as required.

List of Eliminated Nodes EN. Each entry of EN is initialized to $\perp$ on (**10.46**), and is only modified on line (**14.132**). EN is an array with $n-1$ entries, indexed by the nodes in $\mathcal{P} \setminus S$. Here, when a node $(\widehat{N}, \mathtt{T})$ is added to EN, this means the entry of EN corresponding to $\widehat{N}$ is switched to $\mathtt{T}$. Thus, at all times $EN \in [1..n-1] \times \{0,l\}^P$ as required.

Which Neighbor's Know Another Node's Status Report. Parcels of this kind are only added to or removed from DB on lines (**14.134**), (**14.139**), (**14.149**), and (**14.151**). We will now show that for any pair of nodes $\widehat{N}, \widehat{B} \in \mathcal{P} \setminus S$, the data buffer of any node $N \in G$ will have at most one parcel of the form $(\widehat{B}, \widehat{N}, \mathtt{T}')$, from which we conclude that this portion of N's data buffer need hold at most $(n-1)^2$ parcels. To see this, we fix $\widehat{B}$ and $\widehat{N}$ in G and suppose for the sake of contradiction that N's data buffer holds two different parcels $(\widehat{B}, \widehat{N}, \mathtt{T}')$ and $(\widehat{B}, \widehat{N}, \mathtt{T}'')$, for $\mathtt{T}' \neq \mathtt{T}''$. We consider the transmission $\mathtt{T}$ for which this first happens, i.e. without loss of generality, $(\widehat{B}, \widehat{N}, \mathtt{T}')$ is added to DB as on (**14.151**) of $\mathtt{T}$. Since the second part of (**14.151**) is reached, the first part of (**14.151**) must have been satisfied, and since the blacklist is cleared at the end of every transmission (**15.203**), it must be that a broadcast parcel of form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T})$ adding $\widehat{N}$ to N's version of the blacklist was received earlier in transmission $\mathtt{T}$. Notice that necessarily $\widehat{\mathtt{T}} = \mathtt{T}'$, since otherwise line (**14.151**) will not be satisfied. But then since $\mathtt{T}'' \neq \mathtt{T}'$, $(\widehat{B}, \widehat{N}, \mathtt{T}'')$ should have been removed from DB as on (**14.139**) of transmission $\mathtt{T}$, yielding the desired contradiction.

Adding these three contributions, we see that that DB requires at most n^2 parcels, as required.

Outgoing Signature Buffers SIG (**9.17**). Each outgoing signature buffer is initially empty (**10.54**), and they are only modified on (**12.48-49**), (**14.128**), (**14.133**), (**14.141**), and (**14.146**). The first of these increments $SIG[3]$ by at most $2n$, increments $SIG[1]$, and $SIG[p]$ by at most 1, and increments $SIG[2]$ by at most $2n$, and the latter four lines all reset all entries of SIG to $\perp$. Since our protocol is only intended to run polynomially-long (in n), each entry of SIG is polynomial in n, as required.

Incoming Signature Buffers SIG (**9.32**). Each incoming signature buffer is initially empty (**10.48**), and they are only modified on (**12.74-75**), (**14.128**), (**14.133**), (**14.141**), and (**14.146**). The first of these increments $SIG[3]$ by at most $2n$, $SIG[1]$ and $SIG[p]$ by at most 1, and $SIG[2]$ by at most $2n$, and the latter four lines all reset all entries of SIG to $\perp$. Since our protocol is only intended to run polynomially-long (in n), each entry of SIG is polynomial in n, as required.

Copy of Packets Buffer $COPY$ (**10.60**). $COPY$ is first filled on (**10.74**)/(**15.214**), with a copy of every packet corresponding to the first codeword. The only place it is modified after this is on (**15.214**), where the old copies are first deleted and then replaced with new ones.

Sender's Broadcast Buffer BB. In contrast to an internal node's broadcast buffer, the only thing the sender's broadcast buffer holds is the *Start of Transmission* broadcast (**15.200**) and the information that a node should be *removed* from the blacklist, see (**14.165**). Notice that at the outset of the protocol, BB only holds the Start of Transmission broadcast, which is comprised by only $\Omega_1 = (0,0,0,0)$ (**10.72-73**). After this, the only changes made to BB appear on lines (**14.165**), (**15.171**), (**15.199**), and (**15.200**). Notice that for every transmission, necessarily either (**15.171**) or (**15.199**) will be reached, and hence at any time of any transmission $\mathtt{T}$, BB contains parcels corresponding to at most one *Start of Transmission* broadcast, and whatever parcels were added to BB so far in $\mathtt{T}$. By investigating line (**15.200**) and using Lemma 10.4, the former requires at most $2n$ parcels, and by the comment on (**14.156**), the latter requires at most n parcels (**14.165**). Therefore, the sender's broadcast buffer requires at most $3n$ parcels, as required.

Sender's Data Buffer DB, Eliminated List EN, and Blacklist BL (**10.62-64**). We will show that the sender's DB needs to hold at most $n^3 + n^2 + n$ parcels at any time, and that the blacklist and list of eliminated nodes need at most n parcels each. Notice that every entry of DB is initialized to $\perp$ on (**10.73**), after which modifications to DB occur only on lines (**14.158**), (**14.160**), (**14.162**), (**14.166**), (**15.170**), (**15.171**), (**15.187**), (**15.188**), (**15.191**), (**15.194**), (**15.197**), and (**15.199**). The sender's data buffer holds eight different kinds of information: end of transmission parcel $\Theta_{\mathtt{T}}$, status report parcels, the participating list for up to $n-1$ failed transmissions, the reason for failure for up to $n-1$ failed transmissions, its own status reports for up to $n-1$ failed transmissions, the blacklist, list of eliminated nodes, and for each neighbor $B \in G$, a list of nodes $\widehat{N} \in G$ for which $\widehat{B}$ knows the complete status report (see item 4 on line (**13.115**)).

1. End of Transmission Parcel $\Theta_{\mathtt{T}}$. Modifications to this occur only on lines (**14.158**), (**15.171**), and (**15.199**). Every transmission, the inforgibility of the signature scheme and the comment on line (**14.156**) guarantee that the sender will add $\Theta_{\mathtt{T}}$ to DB as on (**14.158**) at most once. Meanwhile, for every transmission, either (**15.171**) or (**15.199**) will be reached exactly once. Therefore, there is at most one End of Transmission parcel in DB at any time.
2. Blacklist BL. We show that BL consists of at most n parcels at any time. More specifically, we will show that BL lives in the domain $[1..n] \times \{0,1\}^{O(\log n)}$, i.e. an array with n slots indexed by each $N \in G$, with each slot holding $\perp$ (if the corresponding node is not on the blacklist) or the index of the transmission in which the corresponding node was most recently added to the blacklist. To see this, notice that modifications to the blacklist occur only on lines (**14.166**), (**15.171**), and (**15.188**). "Removing" a node $\widehat{N}$ from BL as on (**14.166**) means changing the entry indexed by $\widehat{N}$ to $\perp$. "Clearing" the blacklist as on (**15.171**) means making every entry of the array equal to $\perp$. Finally, "adding" a node to the blacklist as on (**15.188**) means switching the entry indexed by $\widehat{N}$ to be the index of the current transmission.
3. Status Report Parcels. Modifications to this occur only on lines (**14.162**) and (**15.171**). We show in Lemma 11.3 below that for any node $\widehat{N} \in \mathcal{P} \setminus S$, DB will hold at most $n(n-1)$ status report parcels from $\widehat{N}$ at any time, from which we conclude that DB need hold at most $n(n-1)^2$ status report parcels.

4. Participating Lists. We will view the participating list corresponding to transmission $\mathtt{T}$ as an array $[1..n] \times \{0,1\}^P$, where the array is indexed by the nodes, and an entry corresponding to node $N \in G$ is either the index of the transmission $\mathtt{T}$ (if N participated in $\mathtt{T}$) or $\perp$ otherwise. Therefore, since each participating list consists of n parcels, we can argue that participating lists require at most $n(n-1)$ parcels if we can show that DB need hold at most $n-1$ participating lists at any time. To see this, notice that (**15.187**) is reached only in the case the transmission *failed* (**15.185**), and we showed in Lemma 10.4 that there can be at most $n-1$ failed transmissions before a node is necessarily eliminated and DB is cleared as on (**15.171**).
5. Reason Transmissions Failed. Modifications to this occur only on lines (**15.171**), (**15.191**), (**15.194**), and (**15.197**). Notice that of the latter three, exactly one will be reached if and only if the transmission failed. Also, each one of the three will add at most one parcel to DB. Since DB is cleared any time ***Eliminate Node*** is called as on (**15.171**), we again use Lemma 10.4 to conclude that Reason for Transmission Failures require at most $n-1$ parcels of DB.
6. Sender's Own Status Reports. These are added to DB on lines (**15.191**), (**15.194**), and (**15.197**), and removed from DB on (**15.171**). Notice that of the former three lines, exactly one will be reached if and only if the transmission failed. Also, each one of the three will add at most n parcels to DB. Since DB is cleared any time ***Eliminate Node*** is called as on (**15.171**), we again use Lemma 10.4 to conclude that Reason for Transmission Failures require at most $n(n-1)$ parcels of DB.
7. List of Eliminated Nodes EN. Modifications to this occur only on line (**15.170**). Since EN is viewed as living in $[1..n] \times \{0,1\}^{O(\log n)}$, "adding" a node $\widehat{N}$ to EN means changing the entry indexed by $\widehat{N}$ from $\perp$ to the index of the current transmission. Notice that $EN \in [1..n] \times \{0,1\}^{O(\log n)}$ can be expressed using n parcels.
8. The Label of a Node $\widehat{N}$ Whose Status Report is Known to B. Modifications to this occur only on lines (**14.160**), (**14.166**), and (**15.171**). We show in Lemma 11.5 below that for any pair of nodes $B, \widehat{N} \in \mathcal{P} \setminus S$, DB will hold at most one parcel of the form $(B, \widehat{N}, \mathtt{T}')$ at any time (see e.g. (**14.160**)), from which we conclude that DB need hold at most $(n-1)^2$ parcels of this type.

Adding together these changes, the sender's DB needs to hold at most $n^3 + n^2 + n$ parcels, as required.

We have now shown each of the variables of Figures 9 and 10 have domains as indicated. ■

Lemma 11.3. *For any node $\widehat{N} \in \mathcal{P} \setminus S$, the sender's data buffer will hold at most $n(n-1)$ status report parcels from $\widehat{N}$ at any time. More specifically, let $\{\mathtt{T}_1, \dots, \mathtt{T}_j\}$ denote the set of transmissions for which the sender has at least one status report parcel from $\widehat{N}$. Then $j \le n-1$ and for every $i < j$, the sender has $\widehat{N}$'s complete status report for transmission $\mathtt{T}_i$.*

Proof. We first note that the first sentence follows immediately from the latter two since each status report consists of at most n status report parcels (**14.142-145**). Fix $\widehat{N} \in \mathcal{P} \setminus S$ and let $\{\mathtt{T}_1, \dots, \mathtt{T}_j\}$ be as in the lemma, ordered chronologically. We first show that $j \le n-1$. For the sake of contradiction, suppose $j \ge n$. We first argue that for all $1 \le i \le j$, transmission $\mathtt{T}_i$ necessarily failed. Fix $1 \leq i \leq j$. Since DB contains a status report parcel from $\widehat{N}$ for transmission $\mathtt{T}_i$, it must have been added on (**14.162**) of some transmission $\widehat{\mathtt{T}}$. Therefore, line (**14.161**) must have been satisfied, and in particular, $(\widehat{N}, \mathtt{T}_i)$ must have been on BL during $\widehat{\mathtt{T}}$. Therefore, $(\widehat{N}, \mathtt{T}_i)$ must have been added to BL as on (**15.188**) of transmission $\mathtt{T}_i$, which in turn implies transmission $\mathtt{T}_i$ failed (**15.185**). Therefore, transmission $\mathtt{T}_i$ failed for each $1 \leq i \leq j$.

By Lemma 10.4, there can be at most $n-1$ failed transmissions before a node is eliminated as on (**15.169-177**). Since $j \geq n$, considering failed transmissions $\{\mathtt{T}_2, \ldots, \mathtt{T}_j\}$, there must have been a transmission $\mathtt{T}_2 \leq \mathtt{T} \leq \mathtt{T}_j$ such that ***Eliminate N*** (**15.169**) was entered in transmission $\mathtt{T}$. We first argue that $\mathtt{T} < \mathtt{T}_j$ as follows. If $\mathtt{T} = \mathtt{T}_j$, then $(\widehat{N}, \mathtt{T}_j)$ would *not* be added to BL as on (**15.188**) (once the protocol enters (**15.169**), it halts until the end of the transmission (**15.177**), thus skipping (**15.188**)). But then (**14.161**) of any transmission after $\mathtt{T}_j$ cannot be satisfied for any status report parcel corresponding to $\mathtt{T}_j$, and hence none of $\widehat{N}$'s status report parcels corresponding to $\mathtt{T}_j$ could be added to DB after transmission $\mathtt{T}_j$. Similarly, none of $\widehat{N}$'s status report parcels corresponding to $\mathtt{T}_j$ can be added to DB before or during transmission $\mathtt{T}_j$ by Claim 11.4 below. This then contradicts the fact that at some point in time, DB contains one of $\widehat{N}$'s status report parcels corresponding to $\mathtt{T}_j$.

We now have that for some transmission $\mathtt{T}_1 < \mathtt{T} < \mathtt{T}_j$, ***Eliminate N*** is entered during $\mathtt{T}$. Therefore, all of $\widehat{N}$'s status report parcels for $\mathtt{T}_1$ are removed from DB on (**15.171**) and $(\widehat{N}, \mathtt{T}_1)$ is removed from BL on (**15.171**) of transmission $\mathtt{T} \leq \mathtt{T}_j$. Since $\mathtt{T}_1 < \mathtt{T}$, $(\widehat{N}, \mathtt{T}_1)$ will never be put on BL as on (**15.190**) for any transmission after $\mathtt{T}$, and consequently, (**14.161**) will never be satisfied after $\mathtt{T}$ for any of $\widehat{N}$'s status report parcels from $\mathtt{T}_1$. Therefore, none of $\widehat{N}$'s status report parcels will be put into DB after they are removed on (**15.171**) of $\mathtt{T}$. Meanwhile, by the end of transmission $\mathtt{T} < \mathtt{T}_j$, DB cannot have any of $\widehat{N}$'s status report parcels corresponding to $\mathtt{T}_j$ by Claim 11.4 below. We have now contradicted the assumption that DB simultaneously holds some of $\widehat{N}$'s status report parcels from $\mathtt{T}_1$ and $\mathtt{T}_j$. Thus, $j \leq n-1$, as desired.

We now show that for every $i < j$, the sender has $\widehat{N}$'s *complete* status report for transmission $\mathtt{T}_i$. If $j = 1$, there is nothing to prove. So let $1 < j \leq n-1$, and for the sake of contradiction suppose there is some $i < j$ such that the sender has at least one of $\widehat{N}$'s status report parcels for $\mathtt{T}_i$, but not the entire report. Let $\widehat{\mathtt{T}}_i$ denote the transmission that the status report parcel corresponding to $\mathtt{T}_i$ was added to DB as on (**14.162**), and let $\widehat{\mathtt{T}}_{i+1}$ denote the transmission that the parcel corresponding to $\mathtt{T}_{i+1}$ was added to DB as on (**14.162**). Without loss of generality, we suppose that $\widehat{\mathtt{T}}_i \leq \widehat{\mathtt{T}}_{i+1}$. Since (**14.162**) is entered during transmission $\widehat{\mathtt{T}}_i$, it must be that (**14.161**) was satisfied, and in particular $(\widehat{N}, \mathtt{T}_i)$ was on BL during $\widehat{\mathtt{T}}_i$. Similarly, $(\widehat{N}, \mathtt{T}_{i+1})$ was on BL during $\widehat{\mathtt{T}}_{i+1}$. Lemma 11.6 below states that for each $N \in G$, N is on BL at most once, i.e. there is at most one entry of the form $(N, \widehat{\mathtt{T}})$ on BL at any time. Since nodes are only added to BL at the very end each transmission (**15.188**), we may conclude that we have strict inequality: $\widehat{\mathtt{T}}_i < \widehat{\mathtt{T}}_{i+1}$. In particular, $(\widehat{N}, \mathtt{T}_{i+1})$ was *not* on BL at the start of $\widehat{\mathtt{T}}_i$, but $(\widehat{N}, \mathtt{T}_i)$ was. Therefore, $(\widehat{N}, \mathtt{T}_{i+1})$ was added to BL as on (**15.188**) of transmission $\mathtt{T}_{i+1}$, and so $\widehat{N} \in \mathcal{P}_{\mathtt{T}}$ (**15.187-188**). In particular, $\widehat{N}$ is *not* blacklisted by the end of $\mathtt{T}_{i+1}$ (**15.187**). Therefore, there must be some transmission $\mathtt{T} \in [\widehat{\mathtt{T}}_i, \mathtt{T}_{i+1}]$ such that $(\widehat{N}, \mathtt{T}_i)$ is removed from BL as on (**14.166**) or (**15.171**). Both of these lead to a contradiction, as the first implies the sender has $\widehat{N}$'s complete status report for $\mathtt{T}_i$, while the latter implies that all status reports corresponding to $\mathtt{T}_i$ should have been removed from DB. ■

Claim 11.4. *For any $\widehat{N} \in G$ and for any transmission $\mathtt{T}$, the sender's data buffer DB will never hold any of $\widehat{N}$'s status report parcels corresponding to $\mathtt{T}$ before or during transmission $\mathtt{T}$.*

Proof. Let $\widehat{N} \in G$, and for the sake of contradiction, let $\mathtt{T}$ be a transmission such that DB has one of $\widehat{N}$'s status report parcels from $\mathtt{T}$ before or during $\mathtt{T}$. Since status reports are only added to DB on (**14.162**), this implies that there is some transmission $\mathtt{T}' \leq \mathtt{T}$ such that (**14.161**) is satisfied at some point of $\mathtt{T}'$ *before* the ***Prepare Start of Transmission Broadcast*** of transmission $\mathtt{T}'$ is called. This in turn implies that $(\widehat{N}, \mathtt{T})$ was on BL *before* the ***Prepare Start of Transmission Broadcast*** of transmission $\mathtt{T}' \leq \mathtt{T}$ was called. However, this contradicts the fact that the only time $(\widehat{N}, \mathtt{T})$ can be added to BL is during the ***Prepare Start of Transmission Broadcast*** of transmission $\mathtt{T}$ on (**15.188**). ∎

Claim 11.5. *For any pair of nodes $B, \widehat{N} \in G \setminus S$, the sender's data buffer will hold at most one status report parcels of the form $(B, \widehat{N}, \mathtt{T}')$ at any time.*

Proof. Fix $B, \widehat{N} \in G \setminus S$, and suppose for the sake of contradiction that there are two transmissions $\mathtt{T}'$ and $\mathtt{T}''$ such that both $(B, \widehat{N}, \mathtt{T}')$ and $(B, \widehat{N}, \mathtt{T}'')$ are in DB at the same time (note that $\mathtt{T}' \neq \mathtt{T}''$ by the comment on **14.156**). Since parcels of this form are only added to DB on (**14.160**), we suppose without loss of generality that $\mathtt{T}$ is a transmission and $\mathtt{t}$ is a round in $\mathtt{T}$ such that $(B, \widehat{N}, \mathtt{T}'')$ is already in DB when $(B, \widehat{N}, \mathtt{T}')$ is added to DB as on (**14.160**) of round $\mathtt{t}$. Since (**14.160**) is reached, (**14.159**) was satisfied, so in particular $(\widehat{N}, \mathtt{T}')$ is on the sender's (current version of the) blacklist. Similarly, since $(B, \widehat{N}, \mathtt{T}'')$ was (most recently) added to DB as on (**14.160**) of some round $\widehat{\mathtt{t}}$ of some transmission $\widehat{\mathtt{T}} \leq \mathtt{T}$, it must have been that $(\widehat{N}, \mathtt{T}'')$ was on the sender's (version of the) blacklist during round $\widehat{\mathtt{t}}$ of $\widehat{\mathtt{T}}$. By Lemma 11.6 below, since $(\widehat{N}, \mathtt{T}')$ is on the sender's current blacklist (as of round $\mathtt{t}$ of transmission $\mathtt{T}$), and $(\widehat{N}, \mathtt{T}'')$ was on an earlier version of the sender's blacklist, it must be that $(\widehat{N}, \mathtt{T}'')$ was removed from the blacklist at some point between round $\widehat{\mathtt{t}}$ of $\widehat{\mathtt{T}}$ and round $\mathtt{t}$ of $\mathtt{T}$. Notice that nodes are removed from the blacklist only on (**14.166**) and (**15.171**). However, in both of these cases, $(B, \widehat{N}, \mathtt{T}'')$ should have been removed from DB (see (**14.166**) and (**15.171**)), contradicting the fact that it is still in DB when $(B, \widehat{N}, \mathtt{T}')$ is added to DB in round $\mathtt{t}$ of transmission $\mathtt{T}$. ∎

Lemma 11.6. *A node is on at most one blacklist at a time. In other words, whenever a node $(N, \mathtt{T})$ is added to the sender's blacklist as on (***15.188***), we have that $(N, \mathtt{T}') \notin BL$ for any other (earlier) transmission $\mathtt{T}'$. Additionally, if $(N, \mathtt{T}') \in BL$ at any time, then:*

1. *Transmission $\mathtt{T}'$ failed*
2. *No node has been eliminated since $\mathtt{T}'$ to the current time*
3. *The sender has not received N's complete status report corresponding to $\mathtt{T}'$*

Proof. The first statement of the lemma is immediate, since the only place a (node, transmission) pair is added to BL is on (**15.188**), and by the previous line, necessarily any such node is not already on the blacklist. Also, Statement 1) is immediate since (**15.188**) is only reached if the transmission fails (**14.185**). To prove Statements 2) and 3), notice that $(N, \mathtt{T}')$ is only added to the blacklist at the very end of transmission $\mathtt{T}'$ (**15.188**). In particular, if $(N, \mathtt{T}')$ is ever removed from the blacklist during some transmission $\mathtt{T}$[37] as on (**14.166**) or (**15.171**), then $(N, \mathtt{T}')$ can never again appear on the blacklist (as remarked in the footnote, $\mathtt{T} > \mathtt{T}'$, and so at any point during or

[37]If $(N, \mathtt{T}')$ is removed from the blacklist as on (**14.166**) or (**15.171**) of transmission $\mathtt{T}$, then necessarily $\mathtt{T} > \mathtt{T}'$, since $(N, \mathtt{T}')$ can only be added to BL at the very end of a transmission (**15.188**), i.e. lines (**14.166**) and (**14.171**) cannot be reached after line (**15.188**) in the same transmission.

after transmission $\mathtt{T}$, $(N, \mathtt{T}')$ can never again be added to BL as on (**15.188**) since $\mathtt{T}'$ has already passed). Therefore, if during transmission $\mathtt{T}$ a node is eliminated as on (**15.169-177**) or the sender receives N's complete status report of transmission $\mathtt{T}'$ as on (**14.164**), then N will be removed from the blacklist as on (**14.166**) or (**15.171**), at which point $(N, \mathtt{T}')$ can never be added to BL again (since necessarily $\mathtt{T} > \mathtt{T}'$ as remarked in the footnote). This proves Statements 2) and 3). ∎

Lemma 11.7. *For any $(N, \mathtt{T})$ on the sender's blacklist, the sender needs at most n parcels from N in order to have N's complete status report, and subsequently remove N from the blacklist (***14.164-165***).*

Proof. This was proven when discussing the appropriateness of variable domains in Lemma 11.2. ∎

We set the following notation for the remainder of the section. $\mathtt{T}$ will denote a transmission, $G_{\mathtt{T}}$ will denote the set of non-eliminated nodes at the start of $\mathtt{T}$, $\mathcal{P}_{\mathtt{T}}$ will denote the participating list for $\mathtt{T}$, and $\mathcal{H}_{\mathtt{T}}$ will denote the uncorrupted nodes in the network. If the transmission is clear or unimportant, we suppress the subscripts for convenience, writing instead $G, \mathcal{P}$, and $\mathcal{H}$.

Lemma 11.8. *For any honest node $A \in G$ and any transmission $\mathtt{T}$, A must receive the complete Start of Transmission (SOT) broadcast before it transfers or re-shuffles any packets. Additionally, the signature buffers $SIG_{A,A}$ and SIG^A of any honest node $A \in G$ are always cleared upon receipt of the complete SOT broadcast (and hence before any packets are transferred to/from/within A).*

Proof. Fix an honest node $A \in G$ and a transmission $\mathtt{T}$. If A has not received the full *Start of Transmission* (SOT) broadcast for $\mathtt{T}$ yet, then A will not transfer any packets (**12.59**), (**11.31-33**), (**12.63**) and (**11.35-37**). This means that (**12.63**) will always be satisfied, and hence (**12.78**) can never be reached, and so RR will remain equal to -1 ((**10.50**), and (**15.209**)) so long as no codeword packets have been transferred. This in turn implies (**12.46**) cannot be satisfied before any codeword packets have been transferred. Putting these facts together, the signature buffers cannot change as on (**12.48-50**), (**12.74-75**), (**12.80**), or (**12.82**) before A receives the complete *SOT* broadcast. Also, no packets will be re-shuffled during the call to Re-Shuffle if no packets have moved during the Routing Phase, as the condition statement on (**7.74**) was eventually false in the last round of the previous transmission, and the state of the buffers will not have changed if no packets have been transferred in the current transmission. Therefore, before A has received the complete *SOT* broadcast, no packet movement to/from/within A is possible, and changes to the signature buffers are confined to the ones appearing on lines (**14.128**), (**14.133**), (**14.141**), and (**14.146**), all of which clear the signature buffers.

Suppose now that A has received the full *SOT* broadcast for $\mathtt{T}$. Recall that part of the *SOT* broadcast contains $\Omega_{\mathtt{T}} = (|EN|, |BL|, F, *)$, where EN refers to the eliminated nodes, BL is the sender's current blacklist, F is the number of failed transmissions since the last node was eliminated, and the last coordinate denotes the reason for failure of the previous transmission (in the case it failed) (**15.200**). If $|BL| = 0$, then A will clear all its entries of SIG^A and $SIG_{A,A}$ on (**14.128**). Otherwise, $|BL| > 0$, and N will clear all its entries of SIG^A and $SIG_{A,A}$ when it learns the last blacklisted node on (**14.146**). Therefore, in all cases A's signature buffers are cleared by the time it receives the full *SOT* broadcast, and in particular before it transfers any packets in transmission $\mathtt{T}$. ∎

In order to prove a variant of Lemma 6.13 in terms of the variables used in the (node-controlling + edge-scheduling) adversary protocol, we will need to first re-state and prove variants of Lemmas

6.11, 7.14, and 7.15. We begin with a variant of Lemma 6.11 (the first 5 Statements correspond directly with Lemma 6.11, the others do not, but will be needed later):

Lemma 11.9. *For any honest node $A \in G$ and at all times of any transmission:*

1. *For incoming edge $E(S, A)$, all changes to $SIG^A[3]_{S,A}$ are strictly non-negative. In particular, at all times:*
$$0 \leq SIG^A[3]_{S,A} \tag{29}$$

2. *For outgoing edge $E(A, R)$, all changes to $SIG^A[3]_{A,R}$ are strictly non-negative*[38]*. In particular, at all times:*
$$0 \leq SIG^A[3]_{A,R} \tag{30}$$

3. *For outgoing edges $E(A, B)$, $B \neq R$, all changes to the quantity $(SIG^A[3]_{A,B} - SIG^A[2]_{A,B})$ are strictly non-negative. This remains true even if B is corrupt. In particular, at all times:*
$$0 \leq \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^A[3]_{A,B} - SIG^A[2]_{A,B}) \tag{31}$$

4. *For incoming edges $E(B, A)$, $B \neq S$, all changes to the quantity $(SIG^A[2]_{B,A} - SIG^A[3]_{B,A})$ are strictly non-negative. This remains true even if B is corrupt. In particular, at all times:*
$$0 \leq \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^A[2]_{B,A} - SIG^A[3]_{B,A}) \tag{32}$$

5. *All changes to $SIG_{A,A}$ are strictly non-negative. In particular, at all times:*
$$0 \leq SIG_{A,A} \tag{33}$$

6. *The net decrease in potential at A (due to transferring packets out of A and re-shuffling packets within A's buffers) in any transmission is bounded by A's potential at the start of the transmission, plus A's increase in potential caused by packets transferred into A. In particular:*
$$SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{A,B} \quad \leq \quad (4n^3 - 6n^2) + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{B,A} \tag{34}$$

7. *The number of packets transferred **out** of A in any transmission must be at least as much as the number of packets transferred **into** A during the transmission minus the capacity of A's buffers. In particular:*
$$4n^2 - 8n \geq \sum_{B \in \mathcal{P} \setminus A} (SIG^A[1]_{B,A} - SIG^A[1]_{A,B}) \tag{35}$$

[38] $SIG^A[3]$ along outgoing edges measures the *decrease* in potential as a *positive* quantity. Thus, a positive value for $SIG^A[3]$ along an outgoing edge corresponds to a decrease in non-duplicated potential.

8. *The number of times a packet p corresponding to the current codeword has been transferred* ***out*** *of A during any transmission is bounded by the number of times that packet has been transferred* ***into*** *A. In particular*[39]*:*

$$0 \geq \sum_{B \in \mathcal{P}} (SIG^B[p]_{A,B} - SIG^A[p]_{B,A}) \tag{36}$$

Proof. We prove each inequality separately, using an inductive type argument on a node A's signature buffers. First, note that all signature buffers are cleared at the outset of the protocol (**10.46**), (**10.48**), and (**10.54**). Also, anytime the signature buffers are cleared as on (**14.128**), (**14.133**), (**14.141**), and (**14.146**), then all of the statements (except possible Statement 8, which depends on values from potentially corrupt nodes $B \in G$) will be true. So it remains to check the other places signature buffers can change values ((**12.48-50**), (**12.74-75**), (**12.80**), (**12.82**), and (**7.76**)), and argue inductively that all such changes will preserve the inequalities of Statements 1-7 (Statement 8 will be proven separately). Since all of these lines represent packet movement, they can only be reached if A has received the complete SOT broadcast for the current transmission (Lemma 11.8), and so we may (and do) assume this is the case in each item below. In particular, Lemma 11.8 states that because we are assuming A has received the complete SOT broadcast for transmission `T`, all of A's signature buffers will be cleared before any changes are made to them.

1. Aside from being cleared, in which case (29) is trivially true, the only changes made to $SIG^A[3]_{S,A}$ occur on (**12.75**), where it is clear that all changes are non-negative since H_{GP} is non-negative (Statement 9 of Lemma 7.1 together with Lemma 11.1).

2. Aside from being cleared, in which case (30) is trivially true, the only changes made to $SIG^A[3]_{A,R}$ occur on (**12.49**), where it is clear that all changes are non-negative since H_{FP} is non-negative (Statement 9 of Lemma 7.1 together with Lemma 11.1).

3. Fix $B \in \mathcal{P} \setminus S, A$. Intuitively, this inequality means that considering directed edge $E(A, B)$, the *decrease* in A's potential caused by packet transfers must be greater than or equal to B's *increase*, which is a consequence of Lemma 6.11. Formally, we will track all changes to the relevant values in the pseudo-code and argue that at all times and for any fixed $B \in G$ (honest or corrupt), if A is honest, then $0 \leq SIG^A[3]_{A,B} - SIG^A[2]_{A,B}$. All changes to these values (aside from being cleared) occur only on (**12.48-49**) since here we are considering A's values along *outgoing* edge $E(A, B)$. Notice that H_{FP} cannot change between (**11.08**) of some round and (**12.49**) of the same round. Since lines (**12.48-49**) are only reached if *Verify Signature Two* accepts the signature (otherwise RR is set to $\perp$ on (**12.90**) and hence (**12.45**) will fail), we have that $SIG^A[2]_{A,B}$ changes by at most the value that H_{FP} had on (**12.89**) (see comments on line (**12.88-90**)), and this is the value sent/received on lines (**11.07**) and (**11.11**) and eventually stored on (**12.48**). Meanwhile, when $SIG^A[3]_{A,B}$ changes, for honest nodes it will always be an increase of H_{FP} (**12.49**), and as noted above, this value of H_{FP} is the same as it had on (**12.89**). Therefore, for honest nodes, whenever the relevant values change on (**12.48-49**), the change will respect the inequality $SIG^A[3]_{A,B} - SIG^A[2]_{A,B} \geq H_{FP} - H_{FP} = 0$.

[39]Notice that (36) is the only statement of the Lemma that involves quantities in the *neighbors'* signature buffers (in addition to A's buffers). Since there is no assumption made about the honesty of the neighbor's of A, this may seem problematic. However, we show in the proof that regardless of the honesty of A's neighbors $B \in G$, (36) will be satisfied if A in honest.

4. Fix $B \in \mathcal{P} \setminus S, A$. Intuitively, this inequality means that considering directed edge $E(B, A)$, the *decrease* in B's potential caused by packet transfers must be greater than or equal to A's *increase*, which is a consequence of Lemma 6.11. Formally, we will track all changes to the relevant values in the pseudo-code and argue that at all times and for any fixed $B \in G$ (honest or corrupt), if A is honest, then $0 \leq SIG^A[2]_{B,A} - SIG^A[3]_{B,A}$. All changes to these values (aside from being cleared) occur only on (**12.74-75**) since here we are considering A's values along *incoming* edge $E(B, A)$. When $SIG^A[2]_{B,A}$ changes on (**12.74**), they take on the values sent by B on (**12.60**) and received by A on (**12.62**). However, in order to reach (**12.74**), the call to ***Verify Signature One*** on (**12.69**) must have returned true. In particular, the comments on (**12.84-86**) require that A verify that the change in $SIG^A[2]_{B,A}$ that B sent to A is at least H_{GP} *bigger* than the previous value A had from B. Meanwhile, when $SIG^A[3]_{B,A}$ changes, for honest nodes it will always be an increase of H_{GP} (**12.75**). Therefore, since H_{GP} cannot change between (**12.84**) of some round and (**12.75**) later in the same round for honest nodes, whenever the relevant values change on (**12.74-75**), the change will respect the inequality $SIG^A[2]_{B,A} - SIG^A[3]_{B,A} \geq H_{GP} - H_{GP} = 0$.

5. Intuitively, this inequality says that all changes in potential due to packet re-shuffling should be strictly non-positive ($SIG_{A,A}$ measures potential *drop* as a *positive* quantity), which is a consequence of Lemma 6.11. Formally, all changes made to $SIG_{A,A}$ (aside from being cleared) occur on (**7.76**), where the change is $M + m - 1$. The fact that this quantity is strictly non-negative for honest nodes follows from Claim 6.4.

6. Since the inequality concerns $SIG_{A,A}$ and $SIG[3]$ (along both incoming and outgoing edges), we will focus on changes to these values when a packet is transferred (or re-shuffled). More specifically, we will look at a specific packet p and consider p's affect on A's potential during each of p's stays in A, where a "stay" refers to the time A receives (an instance of) p as on (**12.77**) to the time it sends and gets confirmation of receipt (as in Definition 7.6) for (that instance of) p[40]. We fix p and distinguish between the four possible ways p can "stay" in A:

   (a) The stay is initiated by A receiving p during T and then sending p at some later round of T, and getting confirmation of p's receipt as in Definition 7.6. More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[3]$ as on (**12.75**) and then an increase to some outgoing signature buffer $SIG^A[3]$ as on (**12.49**). Let B denote the edge along which A received p in this stay, and B' denote the edge along which A sent p. Then $SIG^A[3]_{B,A}$ will increase by H_{GP} on (**12.75**) when p is accepted. Let M denote the value of H_{GP} when p is received. The packet p is eventually re-shuffled to the outgoing buffer along $E(A, B')$. Let m denote the value of H_{FP} when (**12.49**) is reached, so that the change to $SIG^A[3]_{A,B'}$ due to sending p is m. By Statement 3 of Claim 7.7 (which remains valid by Lemma 11.1), any packet that is eventually deleted as on (**12.50-51**) will be the flagged packet, and so the packet that is deleted did actually have height m in A's outgoing buffer. In particular, the packet began its stay in an incoming buffer at height M, and was eventually deleted when it had height m in some outgoing buffer. In particular, since $SIG_{A,A}$ accurately tracks changes in potential due to re-shuffling (Statement 1 of Lemma 11.16), we have that during this stay of p, $SIG_{A,A}$ changed by

[40] A given packet p may have multiple stays in A during a single transmission, one for each time A sees p.

$M - m$. Therefore, considering only p's affect on the following terms, we have that:

$$SIG_{A,A} + SIG^A[3]_{A,B'} - SIG^A[3]_{B,A} = (M - m) + m - M = 0 \tag{37}$$

(b) The stay begins at the outset of the protocol, i.e. p started the transmission in one of A's buffers, and the stay ends when p is deleted (after having been sent across an edge) in some round of T. More specifically, there is no incoming signature buffer $SIG^A[3]$ that changes value as on (**12.75**) due to this stay of p, but there is an increase to some outgoing signature buffer $SIG^A[3]$ as on (**12.49**). Using the notation from (a) above with the exception that M denotes the initial height of p in one of A's buffers at the start of T, then considering only p's affect on the following terms, we have that:

$$SIG_{A,A} + SIG^A[3]_{A,B'} = (M - m) + m = M \tag{38}$$

(c) The stay is initiated by A receiving p during T, but p then remains in A through the end of the transmission (either as a normal or a flagged packet). More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[3]$ as on (**12.75**), but there is no outgoing signature buffer $SIG^A[3]$ that changes value as on (**12.49**) due to this stay of p. Using the notation from (a) above with the exception that m denotes the final height of p in one of A's buffers at the end of T, then considering only p's affect on the following terms, we have that:

$$SIG_{A,A} - SIG^A[3]_{B,A} = (M - m) - M = -m \leq 0 \tag{39}$$

(d) The stay begins at the outset of the protocol, i.e. p started the transmission in one of A's buffers, and p remains in A's buffers through the end of the transmission (either as a normal or a flagged packet). More specifically, there is no incoming signature buffer $SIG^A[3]$ that changes value as on (**12.75**) due to this stay of p, and there is no outgoing signature buffer $SIG^A[3]$ that changes value as on (**12.49**) due to this stay of p. Letting M denote the initial height of p in one of A's buffers at the start of T and m the final height of p in one of A's buffers at the end of T, then considering only p's affect on the following terms, we have that:

$$SIG_{A,A} = M - m \leq M \tag{40}$$

We note that the above four cases cover all possibilities by Claim 6.8 (which remains valid since A is honest, and Lemma 11.1). We will now bound $SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{A,B} - SIG^A[3]_{B,A}$ by adding all contributions to $SIG_{A,A}$ and $SIG^A[3]_{A,B'}$ and $SIG^A[3]_{B,A}$ from all stays of all packets and for all adjacent nodes B, B'. Notice that ignoring contributions as in Case (c) will only help our desired equality, and contributions as in Case (a) are zero, so we consider only packet stays as in (38) and (40). Since these contributions to potential correspond to the initial height the packet had in one of A's buffers at the outset of T, the sum over all such contributions cannot exceed A's potential at the outset of T, which for an honest node A is bounded by $2(n-2)2n(2n+1)/2 < 4n^3 - 6n^2$ (see e.g. proof of Claim 6.2).

7. Intuitively, this inequality means that because a node can hold at most $2(n-2)(2n)$ packets at any time, the difference between the number of packets received and the number of packets sent by an honest node will be bounded by $4n^2 - 8n$. Formally, during a transmission T, the

only places the quantities $SIG[1]$ change are on (**12.74**) and (**12.48**). As with the proof of Statement 6 above, we consider the contribution of each packet p's stay in A[41]:

(a) The stay is initiated by A receiving p during T and then sending p at some later round of T, and getting confirmation of p's receipt as in Definition 7.6. More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[1]$ as on (**12.74**) and then an increase to some outgoing signature buffer $SIG^A[1]$ as on (**12.48**). Let B denote the edge along which A received p in this stay, and B' denote the edge along which A sent p. Since A will be verifying that B (respectively B') signed the correct values (see comments on (**12.84-86**) and (**12.88-90**)), we have that $SIG^A[1]_{B,A}$ will increase by 1 on (**12.74**) due to receiving p for the first time, and $SIG^A[1]_{A,B'}$ will increase by 1 when it receives confirmation of receipt for sending p as on (**12.48**). Therefore, considering only p's affect on the following terms, we have that:

$$SIG^A[1]_{B,A} - SIG^A[1]_{A,B'} = 1 - 1 = 0 \tag{41}$$

(b) The stay is initiated by A receiving p during T, but p then remains in A through the end of the transmission (either as a normal or a flagged packet). More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[1]$ as on (**12.74**), but there is no outgoing signature buffer $SIG^A[1]$ that changes value as on (**12.48**) due to this stay of p. Using the notation from (a) above, then considering only p's affect on the following terms, we have that:

$$SIG^A[1]_{B,A} = 1 \tag{42}$$

We note that the above two cases cover all possibilities by Claim 6.8 (which remains valid since A is honest, see Lemma 11.1). We now add all contributions to $SIG^A[1]_{A,B'}$ and $SIG^A[1]_{B,A}$ from all stays of all packets from all neighbors. Notice that the only non-zero contributions come from packets stays as in (42), and these contributions will correspond to packets that are still in A's buffers at the end of the transmission. Since an honest node A can end the transmission with at most $2(n-2)(2n)$ packets, summing over all such contributions results cannot exceed $4n^2 - 8n$, as required.

8. Intuitively, this is saying that an honest node cannot output a packet more times than it inputs the packet (see Claim 6.8). Note that this is the only place in the theorem that depends on status reports *not* originating from A ($SIG^B[p]$ is a status report parcel from B). A priori, there is the danger that a corrupt B can return a faulty status report, thereby framing A. However, because $SIG^B[p]_{A,B}$ includes a valid signature from A, the inforgibility of the signature scheme guarantees that the only way a corrupt node B can frame A in this manner is by reporting *out-dated* signatures. But if A is honest, then $SIG^B[p]_{A,B}$ is strictly increasing in value as the transmission progresses (the only place it changes is (**12.74**), which comes from the value received on (**12.62**), corresponding to the value sent on (**12.60**)), and hence a corrupt B cannot "frame" A by reporting outdated signatures for $SIG^B[p]_{A,B}$; indeed such a course of action only helps the inequality stated in the theorem. Also notice that

[41] Note that necessarily p is a packet corresponding to the current codeword, since packets corresponding to old codewords do not increment $SIG[1]$, see comments on (**11.59-60**) and (**11.11**). Therefore, there are only two cases to consider.

(other than out-dated signatures) the only place B gets valid signatures from A is on (**12.62**), and this value is one higher than the value that A itself is recording (**12.60**) until A updates $SIG^A[p]_{A,B}$ on (**12.48**). We argue in case (b) below, that whenever B has received an updated $SIG^B[p]_{A,B}$ as on (**12.74**) but A has not yet updated $SIG^A[p]_{A,B}$ as on (**12.48**) (and so these two values differ by one), then Case (b) will contribute -1 to the sum in (36), and therefore the difference of $+1$ between $SIG^B[p]_{A,B}$ and $SIG^A[p]_{A,B}$ will exactly cancel. These two facts allow us to argue (36) by using $SIG^A[p]_{A,B}$ instead of $SIG^B[p]_{A,B}$.

Formally, during a transmission $\mathtt{T}$, the only places the quantities $SIG[p]$ change are on (**12.74**) and (**12.48**). As with the proof of Statement 3 above, we consider the contribution of each packet p's stay in A[42]:

(a) The stay is initiated by A receiving p during $\mathtt{T}$ and then sending p at some later round of $\mathtt{T}$, and getting confirmation of p's receipt as in Definition 7.6. More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[p]$ as on (**12.74**) and then an increase to some outgoing signature buffer $SIG^A[p]$ as on (**12.48**). Let B denote the edge along which A received p in this stay, and B' denote the edge along which A sent p. Since A will be verifying that B (respectively B') signed the correct values (see comments on (**12.84-86**) and (**12.88-90**)), we have that $SIG^A[p]_{B,A}$ will increase by 1 on (**12.74**) due to receiving p for the first time, and $SIG^A[p]_{A,B'}$ will increase by 1 when it receives confirmation of receipt for sending p as on (**12.48**). Therefore, considering only p's affect on the following terms, we have that:

$$SIG^A[p]_{A,B'} - SIG^A[p]_{B,A} = 1 - 1 = 0 \tag{43}$$

(b) The stay is initiated by A receiving p during $\mathtt{T}$, but p then remains in A through the end of the transmission (either as a normal or a flagged packet). More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[p]$ as on (**12.74**), but there is no outgoing signature buffer $SIG^A[p]$ that changes value as on (**12.48**) due to this stay of p. Using the notation from (a) above, then considering only p's affect on the following terms, we have that:

$$-SIG^A[1]_{B,A} = -1 \tag{44}$$

We note that the above two cases cover all possibilities by Claim 6.8 (which remains valid since A is honest, see Lemma 11.1). We now add all contributions to $SIG^A[p]_{A,B'}$ and $SIG^A[p]_{B,A}$ from all stays of p from all neighbors on $\mathcal{P}$ (note that it is enough to consider only neighbors on $\mathcal{P}$ by Claim 11.18). Notice that (43) does not contribute anything, so we have that:

$$\sum_{B \in \mathcal{P}} (SIG^A[p]_{A,B} - SIG^A[p]_{B,A}) = -x, \tag{45}$$

where x is the number of times Case (b) occurs. Notice that (36) is interested in $SIG^B[p]_{A,B}$ (as opposed to $SIG^A[p]_{A,B}$). However, since B cannot report values of $SIG^B[p]_{A,B}$ from

[42]Note that necessarily p is a packet corresponding to the current codeword, since packets corresponding to old codewords do not increment $SIG[p]$, see comments on (**12.59-60**) and (**11.11**). Therefore, there are only two cases to consider.

previous transmissions[43], the only inaccurate value that B can report in its status report parcel concerning $SIG^B[p]_{A,B}$ is by using an older value from T. As discussed above, cheating in this manner only serves to help (36). On the other hand, if B does report the valid value for $SIG^B[p]_{A,B}$ (i.e. not outdated), then Lemma 11.17 guarantees that $SIG^B[p]_{A,B} - SIG^A[p]_{A,B} \le 1$, with equality if $SIG^B[p]_{A,B}$ has been updated as on (**12.74**) and $SIG^A[p]_{A,B}$ has not yet been updated after this point as on (**12.48**). Notice that every time this happens, we fall under Case (b) above, and in particular it can happen at most x times (see definition of x above). Therefore:

$$\sum_{B\in\mathcal{P}}(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}) \le x + \sum_{B\in\mathcal{P}}(SIG^A[p]_{A,B} - SIG^A[p]_{B,A}) = x - x = 0, \quad (46)$$

which is (36).

All Statements of the Theorem have now been proven. ■

We now prove a variant of Lemma 7.14.

Lemma 11.10. *Suppose that $A, B \in G$ are both honest nodes, and that in round* t*, B accepts (as in Definition 6.5) a packet from A. Let $O_{A,B}$ denote A's outgoing buffer along $E(A,B)$, and let H denote the height the packet had in $O_{A,B}$ when* ***Send Packet*** *was called in round* t *(***11.20***). Also let $I_{B,A}$ denote B's incoming buffer along $E(A,B)$, and let I denote the height of $I_{B,A}$ at the start of* t*. Let $\Delta\varphi_B$ denote the* ***change*** *in potential caused by this packet transfer, from B's perspective. More specifically, define:*

$$\varphi_B := SIG^B[2]_{A,B} - SIG^B[3]_{A,B} \quad (47)$$

and then $\Delta\varphi_B$ measures the difference between the value of φ_B at the ***end*** *of* t *and the* ***start*** *of* t*. Then:*

$$\Delta\varphi_B \ge H - I - 1 \qquad OR \qquad \Delta\varphi_B \ge H \ \ (\textit{if } B = R) \quad (48)$$

Furthermore, after the packet transfer but before re-shuffling, $I_{B,A}$ will have height $I+1$.

Proof. By definition, B accepts the packet in round t means that (**12.77**) was reached in round t, and hence so was (**12.74-75**). In particular, $SIG^B[3]_{A,B}$ will increase by H_{GP} on (**12.75**) (if $B = R$, then $SIG^B[3]_{A,B}$ will not change on this line- see comment there). By Statements 1 and 2 of Lemma 7.1 (which remain valid since B is honest by Lemma 11.1), $H_{GP} \le I+1$, and hence $SIG^B[3]_{A,B}$ will increase by at most $I+1$. Also, since B had height I at the start of the round, and B accepts a packet on (**12.77**) of round t, B will have $I+1$ packets in I when the re-shuffling phase of round t begins, which is the second statement of the lemma.

Meanwhile, $SIG^B[2]_{A,B}$ will change on (**12.74**) to whatever value B received on (**12.62**) (as sent by A on (**12.60**) earlier in the round). Since A is honest, this value is H_{FP} larger than A's current value in $SIG^A[3]_{A,B}$ (**12.60**). By Lemma 11.17, the value of $SIG^A[3]_{A,B}$ at the start of t equals the value of $SIG^B[2]_{A,B}$ at the *start* (*before* B has accepted the packet) of t. Therefore, the change in $SIG^B[2]_{A,B}$ from the start of the round to the end of the round will be the value of $H_{FP} = H$ when A reached (**12.60**) in round t (by definition of H and Statement 3 of Claim 7.7). Since these are the only places $SIG^B[3]_{A,B}$ and $SIG^B[2]_{A,B}$ change, we have that $\Delta\varphi_B = H - H_{GP} \ge H - I - 1$, as desired (if $B = R$, then $\Delta\varphi_B = H$). ■

[43] We are only interested in packets p corresponding to the current codeword, and all signatures that A provides for $SIG^B[p]_{A,B}$ include the transmission index, so A's honesty plus the inforgibility of the signature scheme imply that B cannot have any valid signatures from A contributing to $SIG^B[p]_{A,B}$ before the current transmission T.

The following is a variant of Lemma 7.15.

Lemma 11.11. *Let $\mathcal{C} = N_1N_2 \dots N_l$ be a path consisting of l **honest** nodes, such that $R = N_l$ and $S \notin \mathcal{C}$. Suppose that in some **non-wasted** round $\mathtt{t}$, all edges $E(N_i, N_{i+1})$, $1 \le i < l$ are active for the entire round. For $1 \le i < l$, let $\Delta\phi$ denote the following changes to SIG_{N_i,N_i} and SIG^{N_i} during round $\mathtt{t}$:*

1. *Changes to φ_{N_i} (see notation of Lemma 11.10),*
2. *Changes to SIG_{N_i,N_i}*

Then if O_{N_1,N_2} denotes N_1's outgoing buffer along $E(N_1, N_2)$, we have:

- *If O_{N_1,N_2} has a flagged packet that has already been accepted by N_2 **before** round $\mathtt{t}$, then:*

$$\Delta\phi \ge O - l + 1 \tag{49}$$

- *Otherwise,*

$$\Delta\phi \ge O - l + 2 \tag{50}$$

where O denotes its height at the outset of $\mathtt{t}$.

Proof. Since A and B are honest, we use Lemma 11.1 and then follow exactly the proof of the analogous claim for the edge-scheduling model (Lemma 7.15). In particular, the exact proof can be followed, using the fact that signature buffers record accurate changes in non-duplicated potential (Statement 1 of Lemma 11.16), and using Lemma 11.9 in place of Lemma 6.11, and Lemma 11.10 in place of Lemma 7.14. ■

Lemma 11.12. *If at any point in any transmission $\mathtt{T}$, the number of blocked rounds is $\beta_{\mathtt{T}}$, then the participating honest nodes of G will have recorded a drop in non-duplicated potential of at least $n(\beta_{\mathtt{T}} - 4n^3)$. More specifically, the following inequality is true:*

$$n(\beta_{\mathtt{T}} - 4n^3) < \sum_{A \in \mathcal{H} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{H} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} (SIG^A[2]_{B,A} - SIG^A[3]_{B,A}) \tag{51}$$

Proof. For every blocked, non-wasted round $\mathtt{t}$, by the *conforming* assumption there exists a chain $\mathcal{C}_{\mathtt{t}}$ connecting the sender and receiver that satisfies the hypothesis of Lemma 11.11. Letting N_1 denote the first node on this chain (not including the sender), the fact that the round was blocked (and not wasted) means that N_1's incoming buffer was full (see Lemma 11.1), and then by Lemma 6.3, so was N_1's outgoing buffer along $E(N_1, N_2)$. Since the length of the chain l is necessarily less than or equal to n, Lemma 11.11 says that the change of $\Delta\phi$ (see notation there) in round $\mathtt{t}$ satisfies:

$$\Delta\phi \ge O_{N_1,N_2} - l + 1 \ge 2n - n + 1 > n \tag{52}$$

Since $\Delta\phi$ only records some of the changes to the signature buffers, we use Lemma 11.9 to argue that the contributions not counted will only help the bound since they are strictly non-negative. Since we are not double counting anywhere, each non-wasted, blocked round will correspond to an increase in $\Delta\phi$ of at least n, which then yields the lemma since the number of wasted rounds is bounded by $4n^3$ (Lemma 10.9). ■

Lemma 11.13. *If there exists $A, B \in G$ such that one of the following inequalities is not true, then either A or B is necessarily corrupt, and furthermore the sender can identify conclusively*[44] *which is corrupt*[45]*:*

$$
\begin{aligned}
&1. \quad SIG^B[2]_{A,B} \le SIG^A[3]_{A,B} + 2n \\
&2. \quad SIG^A[3]_{S,A} - SIG^S[2]_{S,A} \le 2n \\
&3. \quad |SIG^A[1]_{B,A} - SIG^B[1]_{B,A}| \le 1 \qquad \textit{and} \qquad |SIG^A[1]_{A,B} - SIG^B[1]_{A,B}| \le 1
\end{aligned}
\tag{53}
$$

Proof. As in the first paragraph of the proof of Lemma 11.9, we may assume that both A and B have received the full *Start of Transmission* broadcast for `T`, so SIG^A and SIG^B should both be cleared (if A and B are both honest) of its values from the previous transmission before being updated with values corresponding to the current transmission `T`. We prove each Statement separately:

1. That either A or B is necessarily corrupt follows from Lemma 11.17. It remains to show that the sender can identify a node that is necessarily corrupt. We begin by assuming that $SIG^B[2]_{A,B}$ and $SIG^A[3]_{A,B}$ have appropriate signatures corresponding to `T` (otherwise, they either would not have been accepted as a valid status report parcel on (**14.161**), or a node will be eliminated as on **14.163**). We now show that if the inequality in Statement 1 is *not* true for some $A, B \in G$, then A is necessarily corrupt. Notice that if A is honest, then $SIG^A[3]_{A,B}$ is monotone increasing (other than being cleared upon receipt of the *SOT* broadcast, $SIG^A[3]_{A,B}$ is only updated on **12.49**). Similarly, other than being cleared upon receipt of the *SOT* broadcast, $SIG^B[2]_{A,B}$ is only updated on (**12.74**), and tracing this backwards, this comes from the value received on (**12.62**) which in turn was sent on (**12.60**). Therefore, since B cannot forge A's signature (except with negligible probability or in the case A and B are both corrupt and colluding), $SIG^B[2]_{A,B}$ can only take on values A sent B as on (**12.60**). Meanwhile, as mentioned, if A is honest, $SIG^A[3]_{A,B}$ is monotone increasing, and thus an honest A will never send a value for $SIG^A[3]_{A,B}$ on (**12.60**) of some round that is *smaller* than a value it sent for $SIG^A[3]_{A,B}$ on (**12.60**) of some earlier round. Therefore, since the value A is supposed to send B is $H_{FP} \le 2n$ (the inequality follows from Statement 9 of Lemma 7.1 and Lemma 11.1), unless A is corrupt or B has broken the signature scheme, B will never have a signed value from A such that $SIG^B[2]_{A,B} > 2n + SIG^A[3]_{A,B}$. Therefore, if the inequality in the first statement is not satisfied, A is necessarily corrupt (except with negligible probability).

2. That A is necessarily corrupt follows from Lemma 11.17 and the fact that the sender cannot be corrupted by the conforming restriction placed on the adversary.

3. Note that the two statements are redundant, since the second is identical to the first after swapping the terms on the LHS and re-labelling. We therefore only consider the second inequality of Statement 3. That either A or B is necessarily corrupt follows from Lemma 11.17. It remains to show that the sender can identify a node that is necessarily corrupt. As

[44] As long as the adversary does not break the signature scheme, which will happen with all but negligible probability, the sender will never falsely identify an honest node.

[45] The values of the quantities SIG^B and SIG^A all correspond to a common transmission `T` and refer to values the sender has received in the form of status reports for `T` as on (**14.161**).

in the proof of Statement 1 above, we begin by assuming that $SIG^B[1]_{A,B}$ and $SIG^A[1]_{A,B}$ have appropriate signatures corresponding to $\mathtt{T}$ (otherwise, they either would not have been accepted as a valid status report parcel on (**14.161**), or a node will be eliminated as on **14.163**). We now show that if $|SIG^A[1]_{A,B} - SIG^B[1]_{A,B}| > 1$ for some $A, B \in G$, then either A or B is necessarily corrupt, and the sender can identify which one is corrupt.

Notice that the quantities $SIG^B[1]_{A,B}$ and $SIG^A[1]_{A,B}$ include the *round* in which the quantity last changed ((**11.11**) and (**12.60**)). Let $\mathtt{t}_B$ denote the round $SIG^B[1]_{A,B}$ indicates it was last updated (which has been signed by A), and $\mathtt{t}_A$ denote the round $SIG^A[1]_{A,B}$ indicates it was last updated (which has been signed by B. Note that these quantities refer to the values returned to the sender in the form of status report parcels, and node A (respectively B) has signed the entire parcel $SIG^A[1]_{A,B}$ (respectively $SIG^B[1]_{A,B}$), indicating this is indeed the parcel he wishes to commit to as his status report. We assume $|SIG^A[1]_{A,B} - SIG^B[1]_{A,B}| > 1$, and break the proof into the following two cases:

Case 1: $\mathtt{t}_A > \mathtt{t}_B$. We will show that B is corrupt. Notice that the fact that A has a valid signature on $SIG^A[1]_{A,B}$ from B for round $\mathtt{t}_A$ means that (with all but negligible probability that A could forge B's signature, or if A and B are both corrupt, allowing A to forge B's signature) B sent communication as on (**11.11**) of $\mathtt{t}_A$ with the fifth coordinate equal to the value A used for $SIG^A[1]_{A,B}$. In particular, this fifth coordinate represents the value B has stored for $SIG^B[1]_{A,B}$ during $\mathtt{t}_A$. Since $\mathtt{t}_B < \mathtt{t}_A$, B does not update $SIG^B[1]_{A,B}$ from $\mathtt{t}_B$ through the end of $\mathtt{T}$, and hence the value for $SIG^B[1]_{A,B}$ that B returns the sender in its status report should be the same as the value B sent to A on (**11.11**) of round $\mathtt{t}_A$, which as noted above equals the value of $SIG^A[1]_{A,B}$ that A returned in its status report. However, since this is not the case ($SIG^B[1]_{A,B} \neq SIG^A[1]_{A,B}$), B has returned an outdated signature and must be corrupt.

Case 2: $\mathtt{t}_A \leq \mathtt{t}_B$. If $\mathtt{t}_A = \mathtt{t}_B = 0$, i.e. both nodes agree that they did not update their signature buffers along $E(A, B)$ in the entire transmission (except to clear them when they received the SOT broadcast), then necessarily both $SIG^A[1]_{A,B}$ and $SIG^B[1]_{A,B}$ should be set to $\perp$, so if one of them is *not* $\perp$, the node signing the non-$\perp$ value can be eliminated. So assume that one of the nodes has a valid signature from the other for some round in $\mathtt{T}$ (i.e. that $\mathtt{t}_B > 0$). We will show that A is corrupt in a manner similar to showing B was corrupt above. Indeed, since B has a valid signature from A on $SIG^B[1]_{A,B}$ from round $\mathtt{t}_B$, unless A and B are colluding or B has managed to forge A's signature, this value for $SIG^B[1]_{A,B}$ comes from the communication sent by A on (**12.60**). In particular, since $\mathtt{t}_A \leq \mathtt{t}_B$ and A claims he was not able to update $SIG^A[1]_{A,B}$ after round $\mathtt{t}_A$, the value A signed and sent on (**12.60**) should be exactly one l more than the value stored in $SIG^A[1]_{A,B}$ as of line (**11.07**) of round $\mathtt{t}_A$, the latter of which was returned by A in its status report (by definition of $\mathtt{t}_A$ and the inforgibility of the signature scheme). But since $|SIG^A[1]_{A,B} - SIG^B[1]_{A,B}| > 1$, this must not be the case, and hence A is corrupt. ■

Corollary 11.14. *If there exists a node $A \in G$ such that:*

$$4n^3 - 4n^2 < SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B} - SIG^A[3]_{B,A}, \tag{54}$$

then either a node can be eliminated as in Statement 1 of Lemma 11.13 or as in Statement 6 of Lemma 11.9.

Proof. Suppose no node can be eliminated because of Statement 1 of Lemma 11.13, so that for all $B \in G$:

$$SIG^B[2]_{A,B} \leq SIG^A[3]_{A,B} + 2n. \tag{55}$$

Then if (54) is true, we have that:

$$\begin{aligned} 4n^3 - 4n^2 &< SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B} - SIG^A[3]_{B,A} \\ &\leq SIG_{A,A} + 2n^2 + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{A,B} - SIG^A[3]_{B,A} \end{aligned} \tag{56}$$

where the second inequality follows from applying (55) to each term of the sum. Therefore, A can be eliminated by Statement 6 of Lemma 11.9. ■

Corollary 11.15. *In the case a transmission fails as in F2, the increase in network potential due to packet insertions is at most $2nD + 2n^2$. In other words, either there exists a node $A \in G$ such that the sender can eliminate A, or the following inequality is true*[46]*:*

$$\sum_{A \in \mathcal{P} \setminus S} SIG^A[3]_{S,A} < 2nD + 2n^2 \tag{57}$$

Proof. If the inequality in Statement 2 of Lemma 11.13 fails for any node $A \in \mathcal{P} \setminus S$, the sender can immediately eliminate A. So assume that the inequality in Statement 2 of Lemma 11.13 holds for every $A \in \mathcal{P} \setminus S$. The corollary will be a consequence of the following observation:

Observation. If a transmission T fails as in F2, then:

$$\sum_{A \in \mathcal{P} \setminus S} SIG^S[2]_{S,A} < 2nD \tag{58}$$

Proof. Let $\kappa_{\mathtt{T}}$ denote the value that κ had at the end of T. Then formally, a transmission falling under F2 means that $\kappa_{\mathtt{T}}$ is less than D. The structure of this proof will be to first show that for any $A \in \mathcal{P} \setminus S$, anytime $SIG^S[2]_{S,A}$ is updated as on (**12.48**), it will always be the case that $2n * SIG^S[1]_{S,A} \geq SIG^S[2]_{S,A}$ (so that in particular that the final value for $SIG^S[2]_{S,A}$ at the end of T is less than or equal to $2n$ times the final value for $SIG^S[1]_{S,A}$). We will then show that at the end of T: $\sum_{A \in \mathcal{P} \setminus S} SIG^S[1]_{S,A} = \kappa_{\mathtt{T}}$. From these two facts, we will have shown:

$$\sum_{A \in \mathcal{P} \setminus S} SIG^S[2]_{S,A} \leq \sum_{A \in \mathcal{P} \setminus S} 2n * SIG^S[1]_{S,A} = 2n\kappa_{\mathtt{T}} < 2nD \tag{59}$$

as required.

[46]The values of the quantities SIG^A correspond to some transmission T and refer to values the sender has received in the form of status reports for T as on (**14.161**).

The first fact is immediate, since for any $A \in \mathcal{P} \setminus S$, whenever $SIG^S[2]_{S,A}$ is updated as on (**12.48**), the statement on (**12.45**) must have been satisfied, and so the statement on (**12.89**) must have been false. In particular, the change in $SIG^S[1]_{S,A}$ was exactly one, and the change in $SIG^S[2]_{S,A}$ was at most $H_{FP} \leq 2n$, where the inequality comes from Statement 9 of Lemma 7.1 and Lemma 11.1 (see comments on lines (**12.88-90**)). The second fact is also immediate, as κ and $SIG^S[1]_{S,A}$ all start the transmission with value zero (or $\perp$) by lines (**10.54**), (**10.70**), (**15.199**), and (**15.213**), and then κ is incremented by one on line (**12.47**) of the outgoing buffer along some edge $E(S, N)$ if and only if $SIG^S[1]_{S,N}$ is incremented by one as on (**12.48**) (as already argued, changes to $SIG^S[1]_{S,A}$ as on (**12.48**) are always increments of one, see e.g. the comments on lines (**12.88-90**)). □

The corollary now follows immediately from the following string of inequalities:

$$\begin{aligned} 2nD &> \sum_{A \in \mathcal{P} \setminus S} SIG^S[2]_{S,A} \\ &\geq -2n^2 + \sum_{A \in \mathcal{P} \setminus S} SIG^A[3]_{S,A} \end{aligned}$$

where the top inequality is the statement of the Observation and the second inequality comes from applying the inequality in Statement 2 of Lemma 11.13 to each term of the sum. ■

Lemma 11.16. *For any honest node $N \in G$ and for any transmission* $\mathtt{T}$*:*

1. *Upon receipt of the complete Start of Transmission (SOT) broadcast for transmission* $\mathtt{T}$*, $SIG_{N,N}$ will be cleared. After this point through the end of transmission* $\mathtt{T}$*, $SIG_{N,N}$ stores the correct value corresponding to the current transmission* $\mathtt{T}$ *(as listed on* **9.12***).*

2. *Suppose that N transfers at least one packet during* $\mathtt{T}$ *(i.e. N sends or receives at least one packet, as on (***12.60***) or (***12.74-78***)). Then through all transmissions after* $\mathtt{T}$ *until the transmission and round ($\mathtt{T}', \mathtt{t}' \in \mathtt{T}'$) that N next receives the complete SOT transmission for* $\mathtt{T}'$*, one of the following must happen:*

   (a) All of N's signature buffers contain information (i.e. signatures from neighbors) pertaining to $\mathtt{T}$*, or*

   (b) All of N's signature buffers are clear and N's broadcast buffer[47] *contains all of the information that was in the signature buffers at the end of* $\mathtt{T}$*, or*

   (c) $(N, \mathtt{T}, \mathtt{T}')$ is not on the blacklist for transmission $\mathtt{T}'$

3. *If N has received the full SOT broadcast for* $\mathtt{T}$*, then all parcels in N's broadcast buffer*[47] *BB corresponding to some node $\widehat{N}$'s status report are current and correct. More precisely:*

   (a) If $(\widehat{N}, \widehat{\mathtt{T}})$ is on the sender's blacklist, and at any time N has stored a parcel of $\widehat{N}$'s corresponding status report in its broadcast buffer BB, then this parcel will not be deleted until $(\widehat{N}, \widehat{\mathtt{T}})$ is removed from the sender's blacklist.

[47] Or the Data Buffer in the case $N = S$.

(b) *If* $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T}')$ *is a part of the SOT broadcast of transmission* $\mathtt{T}'$*, then upon receipt of this parcel, all of* $\widehat{N}$*'s status report parcels in* N*'s broadcast buffer correspond to transmission* $\mathtt{T}'$ *and are of the form as indicated on (***14.141-144***), where the reason for failure of transmission* $\mathtt{T}'$ *was determined as on (***15.190***), (***15.193***), or (***15.196***).*

4. *If at any time* N *is storing a parcel of the form* $(B, \widehat{N}, \widehat{\mathtt{T}})$ *in its broadcast buffer (indicating* B *knows* $\widehat{N}$*'s complete status report for transmission* $\widehat{\mathtt{T}}$*), then this will not be deleted until* $(\widehat{N}, \widehat{\mathtt{T}})$ *has been removed from the blacklist.*

Proof. Fix an honest $N \in G$ and a transmission $\mathtt{T}$. We prove each Statement separately:

1. The first part of statement 1 is Lemma 11.8. To prove the second part, we track all changes to $SIG_{N,N}$ and show that each change accurately records the value $SIG_{N,N}$ is supposed to hold. The only changes made to $SIG_{N,N}$ after receiving the full *SOT* broadcast occur on lines (**7.76**), (**12.50**), (**12.80**), and (**12.82**). Meanwhile, $SIG_{N,N}$ is *supposed* to track all packet movement that occurs within N's own buffers (i.e. all packet movement except packet transfers). The only places packets move within buffers of N are on lines (**7.89-90**), (**12.50**), (**12.80**), and (**12.82**). By the comments on lines (**12.50**), (**12.53**), (**12.80**), and (**12.82**), it is clear that $SIG_{N,N}$ appropriately tracks changes in potential due to the call to *Fill Gap*, while packet movement as on (**12.53**) does not need to change $SIG_{N,N}$ as packets are swapped, and so there is no net change in potential. In terms of re-shuffling (**7.89-90**), we see that every packet that is re-shuffled causes a change in $SIG_{N,N}$ of $M - m - 1$ (**7.76**). Notice the actual change in potential matches this amount, since a packet is removed from a buffer at height M (**7.90**), reducing the height of that buffer from M to $M - 1$ (a drop in potential of M), and put into a buffer at height $m + 1$, increasing the height of the buffer from m to $m + 1$ (an increase of $m + 1$ to potential).

2. If $N = S$, there is nothing to show, since the sender's signature buffers' information is stored as needed on (**15.191**), (**15.194**), and (**15.197**), and they are then cleared at the end of every transmission on (**15.171**) or (**15.199**). For any $N \neq S$, we show that from the time N receives the full *SOT* broadcast in a transmission $\mathtt{T}$ through the next transmission $\mathtt{T}'$ in which N next hears the full *SOT* broadcast, either all of N's signature buffers contain information from the last time they were updated in some round of $\mathtt{T}$, or they are empty and either this information has already been transferred to N's broadcast buffer or N is not on the blacklist for transmission $\mathtt{T}'$ (this will prove Statement 2). During transmission $\mathtt{T}$, there is nothing to show, as all changes made to any signature buffer over-write earlier changes, so throughout $\mathtt{T}$, the signature buffers will always contain the most current information. It remains to show that between the end of $\mathtt{T}$ and the time N receives the full *SOT* broadcast of transmission $\mathtt{T}'$, the only change that N's signature buffers can make is to be cleared, and this can happen only if either the information contained in them is first transferred to N's broadcast buffer, or if $(N, \mathtt{T}, \mathtt{T}')$ does not appear in the *SOT* broadcast of transmission $\mathtt{T}'$ (and hence the signature information will not be needed anyway). To do this, we list all places in the pseudo-code that call for a change to one of the signature buffers or removing data from the broadcast buffer, and argue that one of these two things must happen. In particular, the only places the signature buffers of N change (after initialization) are: (**12.48-49**), (**12.50**), (**12.74-75**), (**12.80**), (**12.82**), (**14.128**), (**14.133**), (**14.141**), (**14.146**), and (**7.76**). The only place that

information that was once in one of N's signature buffers is removed from the broadcast buffer is (**14.134**).

First notice that because N transfers a packet in transmission $\mathtt{T}$, N must have received the complete *SOT* broadcast for transmission $\mathtt{T}$ (Lemma 11.8). For all rounds of all transmissions between $\mathtt{T}+1$ and the time N receives the full *SOT* broadcast for transmission $\mathtt{T}'$, lines (**12.48-51**), (**12.74-78**), and (**7.76**) will never be reached by N (see Lemma 11.8 and its proof). Similarly, line (**12.80**) will never be reached since (**12.63**) will always be satisfied. Although (**12.82**) may be reached, we argue that it will not change $SIG_{N,N}$ by arguing that for all rounds between $\mathtt{T}+1$ and the time N receives the full *SOT* broadcast for transmission $\mathtt{T}'$, there will never be codeword packets occupying a higher slot than the ghost packet. More precisely, we will show that for all rounds between $\mathtt{T}+1$ and the time N receives the full *SOT* broadcast for transmission $\mathtt{T}'$, either $H_{GP} = \perp$ or $H_{GP} = H_{IN}+1$, and then by Statements 1 and 2 of Lemma 7.1 (together with the fact that N is honest and so we may apply Lemma 11.1), *Fill Gap* on (**12.82**) will not be performed (see comments on that line). That $H_{GP} = \perp$ or $H_{GP} = H_{IN}+1$ for all of these rounds follows from the fact that H_{GP} will be set to $\perp$ at the end of $\mathtt{T}$ (**15.209**), after which it can only be modified on (**12.66**), (**12.72**), (**12.76**), (**12.78**), (**12.80**), or (**12.82**). Notice that all of these set H_{GP} to $\perp$ or $H_{IN}+1$ and that $H_{IN}+1$ cannot change for all rounds between $\mathtt{T}+1$ and the time N receives the full *SOT* broadcast for transmission $\mathtt{T}'$ by Lemma 11.8.

It remains to consider lines (**14.128**), (**14.133**), (**14.141**), (**14.146**), and (**14.134**); the first four clear the signature buffers, and the last clears the broadcast buffer. So it remains to argue that if any of these lines are reached, either the broadcast buffer is storing all of the information that the signature buffers held at the end of $\mathtt{T}$, or $(N, \mathtt{T}, \mathtt{T}')$ cannot appear as part of the *SOT* broadcast of transmission $\mathtt{T}'$. Line (**14.128**) is clearly covered by the latter case, since if a parcel of this form is received in some transmission $\widehat{\mathtt{T}} \in [\mathtt{T}+1..\mathtt{T}']$, then $(N, \mathtt{T})$ is not on the sender's blacklist as of $\widehat{\mathtt{T}} > \mathtt{T}$, and hence $(N, \mathtt{T})$ will never be able to be re-added to the blacklist after this point (see (**15.188**)). Similar reasoning shows that line (**14.146**) is covered by one of these two cases. In particular, if N reaches line (**14.146**) in some transmission $\widehat{\mathtt{T}} \in [\mathtt{T}+1..\mathtt{T}']$, then either N will add the information in its signature buffers into its broadcast buffers as on (**14.142-145**) before reaching (**14.146**), or else N was not on the blacklist as of $\widehat{\mathtt{T}}$, and hence it is impossible for $(N, \mathtt{T}, \mathtt{T}')$ to be a part of the *SOT* broadcast for transmission $\mathtt{T}'$. Now suppose N reaches (**14.133-134**) in some round of a transmission $\widehat{\mathtt{T}} > \mathtt{T}$ indicating that a node $\widehat{N}$ is to be eliminated. In order to reach (**14.133-134**) in transmission $\widehat{\mathtt{T}}$, N must not have known that $\widehat{N}$ was to be eliminated before that point (**14.131**), and since N received the complete *SOT* broadcast of transmission $\mathtt{T}$ (by Lemma 11.8 together with the hypotheses that N is honest and transferred a packet in $\mathtt{T}$), $\widehat{N}$ must have been eliminated in some transmission $\widetilde{\mathtt{T}} \geq \mathtt{T}$. In particular, if $\widetilde{\mathtt{T}} = \mathtt{T}$, then $(N, \mathtt{T})$ can never be added to the blacklist (since (**14.188**) cannot be reached in transmission $\mathtt{T}$ if *Eliminate Node* is reached in that transmission); while if $\widetilde{\mathtt{T}} > \mathtt{T}$, then $(N, \mathtt{T})$ will be cleared from the blacklist as on (**14.171**) (if it was on the blacklist), and as already remarked, $(N, \mathtt{T})$ can never again appear on the blacklist after this.

Now suppose (**14.141**) is reached in some transmission $\widehat{\mathtt{T}} > \mathtt{T}$ and the signature buffers are cleared on this line. Now before line (**14.141**) was reached, by induction, one of the three statements (a), (b), or (c) was true. If (b) or (c) was true, then changes made on (**14.141**) will not affect the fact that (b) or (c) will remain true. Therefore, assume that we are in case (a) before reaching (**14.141**), i.e. that when (**14.141**) is reached in transmission $\widehat{\mathtt{T}}$, N's signature buffers contain the information that they had at the end of $\mathtt{T}$. Since (**14.141**) was reached, it must have been that $(N, \widetilde{\mathtt{T}}, \widehat{\mathtt{T}})$ was received on (**14.137**) as part of the *SOT* broadcast for transmission $\widehat{\mathtt{T}}$, for some $\widetilde{\mathtt{T}}$. We first argue $\widetilde{\mathtt{T}} \geq \mathtt{T}$. To see this, since N is honest, it will not transfer any packets in $\mathtt{T}$ if it is on its own version of the blacklist ((**11.31-33**) and (**11.35-37**)). Since we know that N *did* transfer packets in transmission $\mathtt{T}$ (by hypothesis), and also N received the full *SOT* broadcast of that same transmission (Lemma 11.8), either N was not on the blacklist as of the start of transmission $\mathtt{T}$, or N received information as on (**14.147**) indicating N could be removed from the blacklist. Both of these cases imply that by the end of $\mathtt{T}$, $(N, \widetilde{\mathtt{T}})$ can never be on the blacklist for any $\widetilde{\mathtt{T}} < \mathtt{T}$. Thus, $\widetilde{\mathtt{T}} \geq \mathtt{T}$, as claimed. Since we are assuming case (a), if $\widetilde{\mathtt{T}} = \mathtt{T}$, then (**14.141**) will *not* be satisfied. On the other hand, if $\widetilde{\mathtt{T}} > \mathtt{T}$, then N has appeared on the blacklist for some transmission *after* $\mathtt{T}$, and then Lemma 11.6 guarantees that $(N, \mathtt{T})$ is not on the blacklist as of $\widehat{\mathtt{T}} > \mathtt{T}$, which as noted above implies $(N, \mathtt{T}, \mathtt{T}')$ cannot be part of the *SOT* broadcast of transmission $\mathtt{T}'$.

3. For Statement (a), we track all the times parcels are removed from N's broadcast buffer BB, and ensure that if ever N removes a status report parcel belonging to $\widehat{N}$ for some transmission $\widehat{\mathtt{T}}$, then $(\widehat{N}, \widehat{\mathtt{T}})$ is no longer on the sender's blacklist. If $N = S$, notice the only place that information concerning other nodes' status report parcels is removed from the sender's data buffer is (**15.171**), and at this point $\widehat{N}$ is not on the blacklist since the blacklist is cleared on this same line.

   If $N \neq S$, changes to BB occur only on lines (**14.134**), (**14.139**), (**14.149**), (**14.142-145**), and (**14.154**). The former three lines *remove* things from BB, while the latter lines *add* things to BB. In terms of statement (a), we must ensure whenever one of the former three lines is reached, there will never be a status report parcel from $\widehat{N}$ and corresponding to transmission $\widehat{\mathtt{T}}$ that is removed from BB if $(\widehat{N}, \widehat{\mathtt{T}})$ is on the blacklist. Looking first at line (**14.134**), suppose that N reaches line (**14.134**) in some transmission $\widetilde{\mathtt{T}} \geq \mathtt{T}$. If $(\widehat{N}, \widehat{\mathtt{T}}, \widetilde{\mathtt{T}})$ was *not* a part of the *SOT* broadcast of transmission $\widetilde{\mathtt{T}}$, then there is nothing to show (since $\widehat{N}$ is not on the blacklist as of the outset of $\widetilde{\mathtt{T}}$). So suppose that $(\widehat{N}, \widehat{\mathtt{T}}, \widetilde{\mathtt{T}})$ was a part of the *SOT* broadcast of transmission $\widetilde{\mathtt{T}}$. Since reaching line (**14.134**) requires that N has newly learned that a node has been added to EN (**14.131**), let N' denote this node, and let $\mathtt{T}'$ denote the round that N' was eliminated from the network as on (**15.170**). First note that necessarily $\mathtt{T}' < \widehat{\mathtt{T}}$. After all, the blacklist will be cleared on line (**15.171**) of round $\mathtt{T}'$, and hence if $(\widehat{N}, \widehat{\mathtt{T}})$ is still on the blacklist as of the outset of $\widetilde{\mathtt{T}}$, it must have been added afterwards. We now argue that because $\mathtt{T}' < \widehat{\mathtt{T}}$, the priority rules of transferring broadcast information will dictate that all honest nodes will necessarily learn N' has been eliminated *before* they learn that $(\widehat{N}, \widehat{\mathtt{T}})$ is on the blacklist. From this, we will conclude that when N reaches (**14.134**) in transmission $\widetilde{\mathtt{T}}$ and learns that N' should be eliminated, that N has not yet learned that $(\widehat{N}, \widehat{\mathtt{T}})$ is on the blacklist, and hence N's broadcast buffer will not be storing any of $\widehat{N}$'s status report parcels for $\widehat{\mathtt{T}}$ (**14.152**).

   It remains to show that any honest node $A \in G$ will learn that N' has been eliminated *before* they learn $(\widehat{N}, \widehat{\mathtt{T}})$ is on the blacklist. So fix an honest node $A \in G$. Suppose A first learns $(\widehat{N}, \widehat{\mathtt{T}})$ is on the blacklist via a parcel of the form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{X})$ that it received as on (**14.137**) of transmission $\mathtt{X}$. Clearly, $\mathtt{X} > \widehat{\mathtt{T}}$, since $(\widehat{N}, \widehat{\mathtt{T}})$ can only be put on the blacklist at the very end of transmission $\widehat{\mathtt{T}}$. Therefore, since $\mathtt{T}' < \widehat{\mathtt{T}} < \mathtt{X}$, we have that $(N', \mathtt{X})$ will be a part of the *SOT* broadcast for transmission $\mathtt{X}$, indicating that N' has been eliminated (**15.200**). Since A is honest, it will therefore receive $(N', \mathtt{X})$ *before* it receives $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{X})$ (see priority rules for receiving broadcast parcels, (**13.110**) and (**13.115**))

We next consider when status report parcels are removed from BB as on (**14.139**). In this case, N has received a *SOT* broadcast parcel of form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T}')$ (**14.137**), and N is removing from BB all of $\widehat{N}$'s status report parcels corresponding to transmissions *other* than $\widehat{\mathtt{T}}$. First note that Lemma 11.6 guarantees that $\widehat{N}$ is on at most one blacklist at any time. Since N received a *SOT* parcel of the form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T}')$ during transmission $\mathtt{T}'$, it must be that $(\widehat{N}, \widehat{\mathtt{T}})$ was on the sender's blacklist at the outset of $\mathtt{T}'$, and since nothing can be added to the blacklist until the very end of a transmission (**15.188**), only $(\widehat{N}, \widehat{\mathtt{T}})$ can be on the sender's blacklist at the outset of $\mathtt{T}'$. This case is now settled, as we have shown that N does not remove any of the status report parcels from $\widehat{N}$ corresponding to $\widehat{\mathtt{T}}$ on (**14.139**), and this is the only transmission for which $\widehat{N}$ can be on the blacklist (at least through $\mathtt{T}'$).

To complete Statement (a), it remains to consider line (**14.149**). But this is immediate, as if the sender at any time removes $(\widehat{N}, \widehat{\mathtt{T}})$ from the blacklist, then it can never again be re-added (since nodes are added to the blacklist at the very *end* of a transmission (**15.188**), they are not removed as on (**14.166**) or (**15.171**) until at least the next transmission, at which point the same $(node, transmission)$ pair $(\widehat{N}, \widehat{\mathtt{T}})$ can never again be added to the blacklist as on (**15.188**) since $\widehat{\mathtt{T}}$ has already passed). Therefore, when N reaches (**14.149**), if the items deleted from BB correspond to $\widehat{N}$, then N must have received a broadcast parcel of form $(\widehat{N}, 0, \mathtt{T})$ as on (**14.147**), indicating that $\widehat{N}$ was no longer on the blacklist. Consequently, the status parcels deleted will never again be needed since $(\widehat{N}, \widehat{\mathtt{T}})$ can never again be on the blacklist.

Part (a) of Statement 3 of the lemma (now proven) states that no status report parcel still needed by the sender will ever be deleted from a node's broadcast buffer. Part (b) states that a node's broadcast buffer will not hold extraneous status report parcels, i.e. status reports corresponding to multiple transmissions for the same node. This is immediate, since whenever a node N learns a node $(\widehat{N}, \mathtt{T}')$ is on the blacklist as on (**14.137**), then N will immediately delete all of its status report parcels from $\widehat{N}$ corresponding to transmissions other than $\mathtt{T}'$ (**14.139**). The fact that the stored parcels have the correct information (i.e. that they address the appropriate reason for failure as on (**14.142-145**)) follows from the fact that N will only initially store a status report parcel if it contains the correct information (**14.153**).

4. There are three lines on which the broadcast parcels of the kind relevant to Statement 4) are removed from N's broadcast buffer: (**14.134**), (**14.139**), and (**14.149**). We consider each of these three lines. Suppose first that the parcel $(B, \widehat{N}, \widehat{\mathtt{T}})$ is removed from N's broadcast buffer as on line (**14.134**) of some transmission $\mathtt{T}'$. In particular, N learns for the first time in the *SOT* broadcast of transmission $\mathtt{T}'$ that some node $\widetilde{N}$ has been eliminated. Let $\widetilde{\mathtt{T}}$ denote the transmission that the sender eliminated this node (as on (**15.169-177**)). If $\widetilde{\mathtt{T}} > \widehat{\mathtt{T}}$, then $(\widehat{N}, \widehat{\mathtt{T}})$ will be cleared from the blacklist on line (**15.171**) of $\widetilde{\mathtt{T}}$, and hence when $(B, \widehat{N}, \widehat{\mathtt{T}})$ is removed from N's broadcast buffer in transmission $\mathtt{T}' > \widetilde{\mathtt{T}}$, $(\widehat{N}, \widehat{\mathtt{T}})$ will no longer be on the blacklist, as required. Therefore, assume $\widetilde{\mathtt{T}} < \widehat{\mathtt{T}}$ (equality here is impossible since lines (**15.169-177**) and (**15.188**) can never both be reached in a single transmission, see e.g. (**15.177**)). Let $\mathtt{X}$ denote the transmission in which N first learned that $(\widehat{N}, \widehat{\mathtt{T}})$ was on the blacklist, i.e. N received a

parcel of the form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{X})$ on (**14.137**) of transmission X. Clearly, $\mathtt{X} > \widehat{\mathtt{T}}$, since $(\widehat{N}, \widehat{\mathtt{T}})$ can only be added to the blacklist at the end of $\widehat{\mathtt{T}}$ (**15.188**). Also, $\mathtt{X} \leq \mathtt{T}'$, since by hypothesis a parcel of the form $(B, \widehat{N}, \widehat{\mathtt{T}})$ is removed from N's broadcast buffer on line (**14.134**) of $\mathtt{T}'$, and this parcel can only have been added to N's broadcast buffer in the first place if N already knew that $(\widehat{N}, \widehat{\mathtt{T}})$ was blacklisted (**14.151**). Lastly, $\mathtt{X} \geq \mathtt{T}'$, since $\widetilde{\mathtt{T}} < \widehat{\mathtt{T}}$ implies that $\widetilde{N}$ was eliminated *before* $(\widehat{N}, \widehat{\mathtt{T}})$ was added to the blacklist, and therefore by the priorities of sending/receiving broadcast parcels ((**13.110**) and (**13.115**)), we have that an honest N will learn that $\widetilde{N}$ has been eliminated *before* it will learn that $(\widehat{N}, \widehat{\mathtt{T}})$ is on the blacklist. Combining these inequalities shows that $\mathtt{X} \geq \mathtt{T}'$ and $\mathtt{X} \leq \mathtt{T}'$, so $\mathtt{X} = \mathtt{T}'$. But this implies that when (**14.134**) is reached in $\mathtt{T}'$, N does not yet know that $(\widehat{N}, \widehat{\mathtt{T}})$ is on the blacklist, and consequently the parcel $(B, \widehat{N}, \widehat{\mathtt{T}})$ cannot yet be stored in N's broadcast buffer, which contradicts the fact that it was removed on (**14.134**) of $\mathtt{T}'$. Therefore, whenever (**14.134**) is reached, either $(\widehat{N}, \widehat{\mathtt{T}})$ will no longer be on the blacklist, or there will be no parcels of the form $(B, \widehat{N}, \widehat{\mathtt{T}})$ that are removed.

Suppose now that the parcel $(B, \widehat{N}, \widehat{\mathtt{T}})$ is removed from N's broadcast buffer as on line (**14.139**) or (**14.149**) of some transmission $\mathtt{T}'$. In either case, by looking at the comments on these lines together with Lemma 11.6, $(\widehat{N}, \widehat{\mathtt{T}})$ has already been removed from the blacklist if a parcel of the form $(B, \widehat{N}, \widehat{\mathtt{T}})$ is removed on either of these lines. ■

Lemma 11.17. *If $A, B \in G$ are honest (not corrupt), in any transmission* T *for which both A and B have received the full SOT broadcast:*

1. *Between the time B accepts a packet from A on line (**12.77**) through the time A gets confirmation of receipt (see Definition 7.6) for it as on (**12.50**), we have:*
   - $SIG^B[1]_{A,B} = 1 + SIG^A[1]_{A,B}$[48]
   - $SIG^B[p]_{A,B} = 1 + SIG^A[p]_{A,B}$[48]
   - $SIG^B[2]_{A,B} = M + SIG^A[3]_{A,B}$, *where M is the value of H_{FP} on (**12.60**) (according to A's view) in the same round in which (**12.77**) was reached by B*
   - $SIG^B[3]_{A,B} = m + SIG^A[2]_{A,B}$, *where m is the value of H_{GP} on (**12.75**) (according to B's view) in the same round in which (**12.77**) was reached by B*
2. *At all other times, we have that $SIG^B[1]_{A,B} = SIG^A[1]_{A,B}$, $SIG^B[2]_{A,B} = SIG^A[3]_{A,B}$, $SIG^B[3]_{A,B} = SIG^A[2]_{A,B}$, and $SIG^B[p]_{A,B} = SIG^A[p]_{A,B}$ for each packet p that is part of the current codeword.*

Proof. The structure of the proof will be as follows. We begin by observing all signature buffers are initially empty (**10.48**) and (**10.54**), and that for any transmission T, both SIG^A and SIG^B are cleared before any packets are transferred (Lemma 11.8). We will then focus on a single transmission for which A and B have both received the full SOT broadcast, and prove that all changes made to SIG^A and SIG^B during this transmission (after the buffers are cleared upon receipt of the SOT broadcast) respect the relationships in the lemma. Since the only changes occur on lines (**12.48-49**) and (**12.74-75**), it will be enough to consider only these 4 lines. Furthermore, if lines (**12.48-49**) were reached x times by A in the transmission, and lines (**12.74-75**) were reached y times by B, then:

[48]If the packet accepted corresponds to an old codeword, then $SIG^B[1]_{A,B} = SIG^A[1]_{A,B}$ and $SIG^B[p]_{A,B} = SIG^A[1]_{A,B} = \perp$.

(a) Either $y = x$ or $y = x + 1$,

(b) Neither set of lines can be reached twice consecutively (without the other set being reached in between)

(c) Lines (**12.74-75**) are necessarily reached *before* lines (**12.48-49**) (i.e. in any transmission, necessarily y will change from zero to 1 *before* x does).

Notice that the the top statement follows from the second two statements, so we will only prove them below.

We first prove the three statements above. We first define x more precisely: x begins each transmission set to zero, and increments by one every time line 50 is reached (just *after* A's signature buffers are updated on lines (**12.48-49**)). Also, define y to begin each transmission equal to zero, and to increment by one when line (**12.74**) is reached (just *before* B's signature buffers are updated on lines (**12.74-75**)). Statement (c) is immediate, since RR begins every round equal to -1 (lines (**10.50**) and (**15.209**)), and can only be changed to a higher index on (**12.78**). Therefore, (**12.46**) can never be satisfied before (**12.78**) is reached, which implies (**12.48**) is never reached before (**12.74**) is. We now prove Statement (b). Suppose lines (**12.48-49**) are reached in some round $\mathtt{t}$. Notice since we are in round $\mathtt{t}$ when this happens, and because RR can never have a higher index than the current round index, and the most recent round RR could have been set is the previous round, we have that B's value for RR (and the one A is using on the comparison on (**12.45-46**)) is at most $\mathtt{t} - 1$. Also, H_{FP} and FR will be set to $\perp$ on (**12.51**) of $\mathtt{t}$. If FR ever changes to a non-$\perp$ value after this, it can only happen on (**12.56**), and so the value it takes must be at least $\mathtt{t}$. Therefore, if at any time after $\mathtt{t}$ we have that $FR \neq \perp$, then if RR has not changed since $\mathtt{t} - 1$, then (**12.46**) can never pass, since $RR \leq \mathtt{t} - 1 < \mathtt{t} \leq FR$. Consequently, (**12.78**) must be reached before (**12.48-49**) can be reached again after round $\mathtt{t}$, and hence so must (**12.74-75**). This shows that (**12.48-49**) can never be reached twice, without (**12.74-75**) being reached in between.

Conversely, suppose lines (**12.74-75**) are reached in some round $\mathtt{t}$. Notice since we are in round $\mathtt{t}$ when this happens, and because FR can never have a higher index than the current round index, we have that A's value for FR (and the one B is using on the comparison on (**12.73**)) is at most $\mathtt{t}$. Also, RR will be set to $\mathtt{t}$ on (**12.78**) of round $\mathtt{t}$, and RR cannot change again until (at some later round) (**12.73**) is satisfied again (or the end of the transmission, in which case their is nothing to show). If line (**12.56**) is NOT reached after (**12.74-75**) of round $\mathtt{t}$, then FR can never increase to a larger round index, so FR will remain at most $\mathtt{t}$. Consequently, line (**12.73**) can never pass, since if B receives the communication from A on line (**12.62**), then by the above comments $RR \geq \mathtt{t} \geq FR$. Consequently, (**12.56**) must be reached before (**12.73**) can be reached again after round $\mathtt{t}$. However, by Statement 3 of Lemma 7.7, (**12.56**) cannot be reached until A receives confirmation of receipt from B (see Definition 7.6), i.e. (**12.56**) can be reached after (**12.74-75**) of round $\mathtt{t}$ only if lines (**12.48-49**) are reached.

We now prove the lemma by using an inductive argument on the following claim:

Claim. *Every time line (***12.74***) is reached (and y is incremented), we have that equalities of Statement 2 of the lemma are true, and between this time and the time line (***12.48***) is reached (or the end of the transmission, whichever comes first), we have that the equalities of the first statement of the lemma are true.*

To prove the base case, notice that before lines (**12.74-75**) are reached for the first time, but after both nodes have received the transmission's SOT broadcast, all entries to both signature buffers are

$\perp$, and so the induction hypothesis is true. Now consider any time in the transmission for which y is incremented by one in some round $\mathtt{t}$ (i.e. line (**12.74**) is reached). Since neither x nor y can change between lines (**11.20**) and (**11.22**), by the induction hypothesis we have that the equalities of the second statement of the lemma are true when A sends the communication as on (**12.60**) of round $\mathtt{t}$. Since A has actually sent $(SIG^A[1]+1, SIG^A[p]+1, SIG^A[3]+H_{FP})$, and these are the quantities that B stores on lines (**12.74-75**), we have that the first statement of the lemma will be true after leaving line (**12.75**) (and in particular the claim remains true). More specifically, letting M denote the value of H_{FP} (respectively letting m denote the value of H_{GP}) when (**12.60**) (respectively (**12.74**)) is reached in round $\mathtt{t}$, we will have that immediately after leaving (**12.75**):

1. $SIG^B[1]_{A,B} = 1 + SIG^A[1]_{A,B}$
2. $SIG^B[p]_{A,B} = 1 + SIG^A[p]_{A,B}$
3. $SIG^B[2]_{A,B} = M + SIG^A[3]_{A,B}$
4. $SIG^B[3]_{A,B} = m + SIG^A[2]_{A,B}$

as required by Statement 1 of the Lemma. By Statement (b) above, either the signature buffers along $E(A,B)$ do not change through the end of the transmission, or the next change necessarily occurs as on (**12.48-49**). In the former case, the Claim certainly remains true. In the latter case, let $\mathtt{t}'$ denote the time that (**12.48**) is next reached. Notice that $\mathtt{t}' > \mathtt{t}$, as Statement (b) above guarantees (**12.48**) is reached *after* (**12.74**), and by examining the pseudo-code, this cannot happen until at least the next round after $\mathtt{t}$. In particular, the values received on (**11.07**) of round $\mathtt{t}'$ necessarily reflect the most recent values of SIG^B (i.e. B's signature buffers have already been updated as on (**12.74-75**) when B sends A the communication on (**12.11**)). Consequently, A will change $SIG^A[1]$, $SIG^A[2]$, and $SIG^A[p]$ to the values B is storing in $SIG^B[1]$, $SIG^B[3]$, and $SIG^B[p]$, respectively. Therefore, the claim (and hence the lemma) will be true provided we can show that when A updates $SIG^A[3]$ as on (**12.49**), that the new value for $SIG^A[3]$ equals the value stored in $SIG^B[2]$. Since before (**12.49**) is reached, we have by the induction hypothesis that $SIG^B[2]_{A,B} = M + SIG^A[3]_{A,B}$, it is enough to show that when $SIG^A[3]$ is updated on (**12.49**), that the value of H_{FP} there equals M. We argue that this by showing H_{FP} will not change from line (**12.60**) of round $\mathtt{t}$ (when M was set to H_{FP}) through line (**12.49**) of round $\mathtt{t}'$. To see this, notice that the only possible places H_{FP} can change during a transmission are lines (**12.51**), (**12.53**), and (**12.56**). Clearly, (**12.51**) cannot be reached between these times, since (**12.49**) is not reached during these times. Also, Statement 3 of Lemma 7.7 implies that (**12.56**) cannot be reached between these times either. Finally, (**12.53**) cannot be reached, since RR will be set to $\mathtt{t}$ on (**12.78**) of round $\mathtt{t}$, and by statement (b), (**12.78**) cannot be reached again until after (**12.49**) is reached in round $\mathtt{t}'$, and hence RR will be equal to $\mathtt{t}$ from (**12.78**) of round $\mathtt{t}$ through (**12.49**) of round $\mathtt{t}'$. Also, FR will not change between these times (also by Statement 3 of Lemma 7.7), and since the only non-$\perp$ value FR is ever set to is the current round as on (**12.56**), we have that $FR \leq \mathtt{t}$. Putting these facts together, we have that for all times between line (**12.60**) of round $\mathtt{t}$ through line (**12.49**) of round $\mathtt{t}'$, either A does not receive RR (in which case $RR = \perp$ when (**12.52**) is reached) or A receives RR, which as noted obeys $RR = \mathtt{t} \geq FR$. In either case, (**12.52**) will fail, and (**12.53**) cannot be reached. ■

Lemma 11.18. *For any transmission* $\mathtt{T}$*, recall that* $\mathcal{P}_{\mathtt{T}}$ *denotes the list of nodes that* ***participated*** *in that transmission, and it is set at the end of each transmission on (***15.187***). For any honest (not* *corrupt) node $A \in G$, during any transmission* $\mathtt{T}$*, A will not exchange any codeword packets with any node that is not put on $\mathcal{P}_{\mathtt{T}}$ at the end of the transmission.*

Proof. Restating the lemma more precisely, for any node N that is NOT put on $\mathcal{P}_{\mathtt{T}}$ as on (**15.187**) and for any honest node $A \in G$, then along (directed) edge $E(A, N)$, A will never reach line (**11.60**), and along (directed) edge $E(N, A)$, A will never reach lines (**12.67-82**). Fix a transmission $\mathtt{T}$ in which (**12.187**) is reached (i.e. a node is not eliminated as on (**15.169-177**) of $\mathtt{T}$), let $N \notin \mathcal{P}_{\mathtt{T}}$ be any node *not* put on $\mathcal{P}_{\mathtt{T}}$ on (**15.187**) of $\mathtt{T}$, and let $A \in G$ be an honest node. Since $N \notin \mathcal{P}_{\mathtt{T}}$, we have that either $N \in EN$ or $N \in BL$ when (**15.187**) is reached. Since no nodes can be *added to EN* or BL from the outset of $\mathtt{T}$ through line (**15.187**) of $\mathtt{T}$, we must have that $N \in EN$ or $N \in BL$ as of either line (**15.188**) or (**15.170**) of the previous transmission. Therefore, either $(N, \mathtt{T})$ or $(N, \mathtt{T}', \mathtt{T})$ is added to the *SOT* broadcast of transmission $\mathtt{T}$ (on (**15.176**) or (**15.200**) of transmission $\mathtt{T} - 1$), indicating N is an eliminated/blacklisted node. If A has not received the full *Start of Transmission* (SOT) broadcast for $\mathtt{T}$ yet, then the lemma is true by Lemma 11.8. If on the other hand A has received the full *SOT* broadcast, then in particular A has received the parcel indicating that N is either eliminated or blacklisted. Thus, by lines (**12.59**), (**11.31-33**), (**12.63**) and (**11.35-37**), A will not transfer any packets with N. ∎

Lemma 11.19. *The receiver's end of transmission broadcast takes at most n rounds to reach the sender. In other words, the sender will have always received the end of transmission broadcast by the time he enters the* ***Prepare Start of Transmission Broadcast*** *segment on (*__11.29__*).*

Proof. By the conforming assumption, for every round $\mathtt{t}$ of every transmission there is a path $\mathsf{P}_{\mathtt{t}}$ between the sender and receiver consisting of edges that are always up and nodes that are not corrupt. We consider the final n rounds of any transmission, and argue that for each round, either the sender already knows the end of transmission parcel Θ, or there is a *new* honest node $N \in G$ that learns Θ for the first time. Since the latter case can happen at most $n - 1$ times (the receiver already knows Θ when there are n rounds remaining, see (**11.28**) and (**15.178-179**)), it must be that the sender has learned Θ by the end of the transmission. Therefore, let $4D - n < \mathtt{t} \leq 4D$ be one of the last n rounds of some transmission. If the sender already knows Θ, then we are done. Otherwise, let $\mathsf{P}_{\mathtt{t}} = N_0 N_1 \ldots N_L$ (here $N_0 = S$ and $N_L = R$) denote the active honest path for round $\mathtt{t}$ that connects the sender and receiver. Since S does not know Θ but R does, there exists some index $0 \leq i < L$ such that N_i does not know Θ but N_{i+1} does know Θ. Since edge $E(N_i, N_{i+1})$ is active and the nodes at both ends are honest (by choice of $\mathsf{P}_{\mathtt{t}}$), node N_{i+1} will send N_i a broadcast parcel on (**11.15**). Looking at the manner in which broadcast parcels are chosen (**13.115**), it must be that N_{i+1} will send Θ to N_i in round $\mathtt{t}$, and hence N_i will learn Θ for the first time, which was to be showed. ∎

Lemma 11.20. *If the receiver has received at least $D - 6n^3$ distinct packets corresponding to the current codeword, he can decode the codeword (except with negligible probability of failure).*

Proof. Fact $1'$ guarantees that if the receiver obtains $D - 6n^3$ distinct packets corresponding to a codeword, then he can decode. Since all codeword packets are signed by the sender to prevent modifying them, the security of the signature scheme guarantees that any properly signed codeword packet the receiver obtains will be legitimate (except with negligible probability of failure). ∎

Lemma 11.21. *For every transmission* $\mathtt{T}$*: $S, R \in \mathcal{P}_{\mathtt{T}}$.*

Proof. The participating list $\mathcal{P}_{\mathtt{T}}$ is set at the end of every transmission on line (**15.187**). By looking at the code there, we must show that $S, R \notin EN \cup BL$ at the end of any transmission. That an honest node can never be identified as corrupt and eliminated is the content of the proof of Theorem 8.1, so $S, R \notin EN$. Since S is never put on the blacklist (**15.188**), it remains to show $R \notin BL$ when (**15.187**) is reached. Since nodes are removed from the blacklist on line (**14.166**) and not put on it again until (**15.188**), it is enough to show that if R is ever placed on the blacklist at the end of some transmission $\mathtt{T}-1$, then it will be removed as on (**14.166**) of transmission $\mathtt{T}$. If R is ever placed on the blacklist, we argue that: 1) R will learn what status report parcels the sender requires of it after at most $2n^2$ rounds; and 2) S will receive all of these parcels by at most $4n^3$ rounds later. Therefore, R will necessarily be removed from the blacklist by round $4n^3 + 2n^2 < 4D$ (since $D \geq 6n^3$), as required. To prove 1), first note that all honest nodes remove the receiver's *end of transmission* parcel for $\mathtt{T}-1$ at the very end of $\mathtt{T}-1$ (**15.203**). Therefore, no honest node will have any *End of Transmission Parcel* in its broadcast buffer at any point during $\mathtt{T}$ *until* one is created for the current transmission on (**15.178-179**). Therefore, for the first n^3 rounds, the sender's SOT broadcast will have top priority in terms of sending/receiving broadcast parcels (**13.115**). Since S and R are connected by an active honest path at each round, we follow the proof as in Lemma 11.19 to argue that for every round between the outset of $\mathtt{T}$ and round n^3, either R has learned the full SOT broadcast, or there is an honest node that is learning a *new* SOT broadcast parcel for the first time. Since there are (at most) n nodes, and the SOT broadcast has at most $2n$ parcels (see proof of Lemma 11.2, and Statement 2 of the Broadcast Buffer therein), it takes at most $2n^2$ rounds for R to receive the full SOT broadcast, and hence to learn it has been blacklisted. This proves 1).

Upon receipt of this information, R adds the necessary information (i.e. its status report) to its broadcast buffer (**14.137-145**). Looking at the proof of Theorem 10.9 and in particular Claim 2 within the proof, edges along the active honest path can take at most $4n^3 < 4D$ rounds to communicate across their edges the broadcast information of priorities 1-6 on lines (**13.115**), and since the receiver is connected to the sender *every* round via some active honest path (by the conforming assumption), its requested status report information will necessarily reach the sender within $4n^3$ rounds, proving 2). ∎

Lemma 11.22. *For any transmission* $\mathtt{T}$*, if* $\mathcal{P}_{\mathtt{T}} = \{S, R\}$*, then the transmission was necessarily successful.*

Proof. $\mathcal{P}_{\mathtt{T}}$ is set on line (**15.187**). Since the only place the sender adds nodes to the blacklist is on (**15.188**), which happens at the very end of each transmission, and because the hypothesis states that every non-eliminated node except for S and R is on the blacklist when line (**15.187**) of transmission $\mathtt{T}$ is reached, it must be the case that transmission $\mathtt{T}$ began with every non-eliminated node on the blacklist, with the possible exception of the receiver (and the sender who is never blacklisted). Since all internal nodes are still blacklisted by the end of the transmission, the sender will never transfer any packets to any node other than R during transmission $\mathtt{T}$ (line (**12.59**) will always fail for any other node, see (**11.31-33**)). Theorem 10.9 indicates there are at most $4n^3$ rounds that are wasted, and since the only edge the sender can ever use to transfer codeword packets during $\mathtt{T}$ is $E(S, R)$, the conforming assumption implies edge $E(S, R)$ is active *every* round of $\mathtt{T}$. We may therefore view the graph as reduced to a single edge connecting S and R (see Lemma 11.18), where there are at least $4D - 4n^3 > 3D$ (non-wasted) rounds per transmission. Since both S and R are honest, correctness is guaranteed as in the edge-scheduling protocol by Lemma 11.1. In particular, the transmission will necessarily be successful. ∎

Lemma 11.23. *No honest node will accept more than one distinct parcel (per node $\widehat{N}$ per transmission) indicating that $\widehat{N}$ should be removed from the blacklist.*

Proof. Line (**13.110**) guarantees that any node A will only accept the parcel if it has already received the sender's *start of transmission* broadcast corresponding to the current transmission. In particular, this means that A has received an updated blacklist (and a list of eliminated nodes) before it accepts any removals from the blacklist. Therefore, in some transmission $\mathtt{T}$, if A ever does accept the information that a node $\widehat{N}$ should be removed from the blacklist, then this information will not become out-dated until (if) $\widehat{N}$ is added to the blacklist again, which can happen at the earliest at the very end of transmission (**15.188**). Therefore, after receiving the information for the first time that $\widehat{N}$ should be removed, the comments on line (**14.123**) will guarantee A will not accept additional blacklist information regarding $\widehat{N}$ until the following transmission, proving the lemma. ■

Lemma 11.24. *For any node $\widehat{N} \in G$, after receiving the complete SOT broadcast, an honest node N will transmit along each edge at most once per transmission the fact that it knows $\widehat{N}$'s complete status report.*

Proof. Each parcel stored in N's broadcast buffer BB is accompanied by a list of which edges the parcel has been successfully transmitted across (see comments on line (**14.123**)). Therefore, as long as the parcel is not deleted from the broadcast buffer, line (**13.115**) guarantees that each parcel of broadcast information will only pass along each edge once, as required. Therefore, it remains to prove the lemma in the case that the relevant broadcast parcel is deleted at some point in a transmission. Fix a transmission $\mathtt{T}$ and an arbitrary $\widehat{N} \in G$. Since broadcast parcels of the relevant type (i.e. that N has $\widehat{N}$'s complete status report) are only removed on (**14.139**) and (**14.149**), we need only consider the case that (**14.149**) is reached in transmission $\mathtt{T}$ (the former line can only be reached as part of the SOT broadcast, and therefore lies outside the hypotheses of the lemma). In particular, we will show that if (**14.149**) deletes from N's broadcast buffer the parcel indicating that N knows $\widehat{N}$'s complete status report, then N will never again add a parcel of this form to its broadcast buffer (as on (**14.155**)) for the remainder of $\mathtt{T}$. But this is immediate, since if N removes this parcel from BB on (**14.149**) of $\mathtt{T}$, then $\widehat{N}$ must have been removed from the blacklist (see (**14.147**)), and since $\widehat{N}$ cannot be re-added to the blacklist until the end of $\mathtt{T}$ (**15.188**), line (**14.152**) (of N's code, with the $\widehat{N}$ that appears there equal to the $\widehat{N}$ used in the present notation) cannot be satisfied for the remainder of $\mathtt{T}$, and hence (**14.155**) cannot be reached. This proves that once the parcel is deleted, it cannot be later added in the same transmission, proving the lemma. ■

12 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[49] 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[50]. 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.

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, and N. Shavit. "*Slide*– The Key to Polynomial End-to-End Communication." *Journal 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 Principles of Distributed Computing, 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] 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.

[6] B. Awerbuch and T. Leighton. "Improved Approximation Algorithms for the Multi-Commodity Flow Problem and Local Competitive Routing in Dynamic Networks." *STOC.* 1994.

[7] 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.

[8] B. Barak, S. Goldberg, and D. Xiao. "Protocols and Lower Bounds for Failure Localization in the Internet." *Proc. of Advances in Cryptology- 27^{th} EUROCRYPT 2008, Springer LNCS 4965, pp. 341-360.* 2008.

[49]In [12], 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.

[50]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 $O(Ln^3 \log n)$.

[9] S. Even, O. Goldreich, and S. Micali. "On-Line/Off-Line Digital Signatures." *J. Cryptology 9(1): pp. 35-67.* 1996.

[10] O. Goldreich. "The Foundations of Cryptography, Basic Applications." Cambridge University Press. 2004.

[11] 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.

[12] E. Kushilevitz, R. Ostrovsky, and A. Rosén. "Log-Space Polynomial End-to-End CommunicIation." *SIAM Journal of Computing 27(6): 1531-1549.* 1998.

[13] S. Micali, C. Peikert, M. Sudan, and D. Wilson. "Optimal Error Correction Against Computationally Bounded Noise." *TCC LNCS 3378, pp. 1-16.* 2005.

[14] S. Rajagopalan and L. Schulman "A Coding Theorem for Distributed Computation." *Proc. 26th STOC, pp. 790-799.* 1994.

[15] C. E. Shannon (Jan. 1949). "Communication in the presence of noise". Proc. Institute of Radio Engineers vol. 37 (1): pp. 10-21.

[16] A. Shamir and Y. Tauman. "Improved Online/Offline Signature Schemes." *CRYPTO 2001, pp. 355-367.* 2001.

[17] L. Schulman. "Coding for interactive communication." *Special issue on Codes and Complexity of the IEEE Transactions on Information Theory 42(6), Part I: pp.1745-1756.* 1996. (Preliminary versions: Proc. 33rd FOCS 724-733, 1992 and Proc. 25th STOC 747-756, 1993).