J. Cryptol. (2014) 27: 636–771
DOI: 10.1007/s00145-013-9157-6

Journal of
CRYPTOLOGY

Authenticated Adversarial Routing*

Yair Amir†
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA
yairamir@cs.jhu.edu

Paul Bunn‡
Google, Mountain View, CA, USA
paulbunn@google.com

Rafail Ostrovsky§
UCLA Departments of Computer Science and Department of Mathematics, Los Angeles, CA, USA
rafail@cs.ucla.edu

Communicated by Oded Goldreich

Received 14 July 2009
Online publication 7 September 2013

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 (with respect to network size) 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

* A preliminary version of this paper appeared in the 6th IACR Theory of Cryptography Conference, pp. 163–182, 2009.

† Part of Y. Amir's work was done while visiting IPAM and supported in part by NSF grant 0430254.

‡ P. Bunn's work was supported in part by NSF grants 0430254, 0716835, 0716389 and 0830803.

§ Part of R. Ostrovsky's work was done while visiting IPAM and supported in part by NSF grants CNS-0430254; CNS-0716835; CNS-0716389; CNS-0830803; CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS-1136174; US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

nodes (though this path may change at every round). Out construction works in the public-key setting and enjoys *optimal transfer rate* and bounded memory per processor (that is polynomial in the network size and does not depend on the amount of traffic). We stress that our protocol assumes no knowledge of which nodes are corrupted nor which path is reliable at any round, and is also fully distributed with nodes making decisions locally, so that they need not know the topology of the network at any time.

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

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

Our result is related to recent work of Barak et al. (Proc. of Advances in Cryptology—27th EUROCRYPT 2008, LNCS, vol. 4965, pp. 341–360, 2008) who studied fault localization in networks assuming a private-key trusted-setup setting. Our work, in contrast, assumes a public-key PKI setup and aims at not only fault localization, but also transmission optimality. Among other things, our work answers one of the open questions posed in the Barak et al. paper regarding fault localization on multiple paths. The use of a public-key setting to achieve strong error-correction results in networks was inspired by the work of Micali et al. (Proc. of 2nd Theory of Cryptography Conf., LNCS, vol. 3378, pp. 1–16, 2005) 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 (Proc. 26th ACM Symp. on Theory of Computing, pp. 790–799, 1994) 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 (Proc. of the 7th ACM Symp. on Principles of Distributed Computing, pp. 131–148, 1988); Awebuch et al. (Proc. of the 30th IEEE Symp. on Foundations of Computer Science, FOCS, 1989); Afek et al. (Proc. of the 11th ACM Symp. on Principles of Distributed Computing, pp. 35–46, 1992); and Afek et al. (J. Algorithms 22:158–186, 1997), 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.

Key words. Network routing, Fault localization, Error-correction, Multi-party computation, End-to-end communication, Communication complexity.

1. Introduction

In this paper we design a routing protocol for an unreliable and dynamically changing synchronous network that is resilient against malicious insiders who may try to destroy and alter messages or disrupt communication in any way. We model the network as a communication graph $G = (V, E)$ where each vertex/node is a processor and each edge is a communication link. We do not assume that the topology of this graph is fixed or known by the nodes. 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. We assume that the capacity of each edge in the network is fixed, so that any edge in the system that is "up" during some round can transmit exactly P bits of information[1] during that round. We assume the sender has bundled the information he wishes to send the receiver into a sequence of "packets" $\{p_1, p_2, \ldots\}$ of size at most P.

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 that were sent by the sender, without duplication or omission.
2. *Throughput.* This measures the number of packets that the receiver has obtained 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" (or "distributed") 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:[2]

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).[3] The adversary can make any arbitrary poly-time computation to maximize interference in routing, so long as it remains *conforming*.

[1] For the protocol of Sect. 4, we require that $P \in \Omega(\log n)$, while the protocol of Sect. 5 requires that $P \in \Omega(k + \log n)$, where k is the security parameter.

[2] 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.

[3] A more general definition of an edge-scheduling adversary would be to allow completely arbitrary edge failures (except that in the limit there must be no permanent cut between the sender and receiver). However, this definition (while more general) greatly complicates the evaluation of a protocol's throughput performance; in particular, our current definition of *throughput rate* cannot be used to compare routing protocols in this generalized setting.

Node-Controlling Adversary. This adversary controls the *nodes* of the network. 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).

Although we could capture the two above forms of adversarial interference by a single adversary, it will be convenient to view these adversaries as distinct, as 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 all communication with nodes that have been identified as corrupt.[4] 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 a

[4] 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.

number of authors, including Afek and Gafni and Rosen [2], Afek and Gafni [1], Awerbuch, Mansour and Shavit [6], Afek, Awebuch, Gafni, Mansour, Rosen, and Shavit [3], and Kushilevitz, Ostrovsky and Rosen [14]. Indeed, our starting point is the *Slide* protocol[5] 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 Leighton [5]) 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 Table 1 of Sect. 1.3 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 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 no fault occurred, nor can they persuade S into believing that the fault occurred on an honest edge. Localizing the fault in their paper relies on cryptographic tools, and in particular the assumption that one-way functions exist. Although utilizing these tools (such as MACs or Signature Schemes) increases communication cost, it is shown by Goldberg, Xiao, Barak, and Redford [12] that the existence of a protocol that is able to securely detect faults (in the presence of a node-controlling adversary) implies the existence of one-way functions, and it is shown in Barak et al. [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.

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

Another paper that addresses routing in the Byzantine setting is the work of Awerbuch, Holmes, Nina-Rotary and Rubens [7], 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 et al. [15] and references therein. It is important to note that this work deals with a graph that has a single "noisy" edge, as modeled by an adversary who can partially control and modify information that crosses that 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 [17], where they show how to correct noisy edges during distributed computation. Their work does not consider actively malicious nodes nor the question of throughput-efficient network routing, and thus is different from our setting. It should also be noted that their work utilizes Schulman's tree-codes [18] that allow length-flexible online error correction. The important difference between our work and that of Schulman is that we do not restrict the amount of malicious activity of corrupt nodes.

1.2. *Subsequent Work*

Subsequent to the submission of this work, there has been a line of research that investigates the feasibility of routing in a network model that is more general than the network model considered here. Specifically, the network model of Bunn and Ostrovsky [10] is susceptible to both edge failures and corruptible nodes (as is the case in the present work), but is different from the model presented in Sect. 3 because of fewer assumptions placed on the overall connectivity of the network: there are no connectivity guarantees, and the network is *asynchronous* (i.e. there is no universal clock to coordinate transmissions across each network link).

Direct comparison of the present results to those in [10] is not possible, as the less restrictive network model of [10] requires a different metric for evaluating performance of protocols within that model. For example, the lack of any assumption on network connectivity means that all edges of the network may be inactive at all times, and thus all protocols would achieve zero throughput. Instead, *competitive analysis* is used in [10] to compare a given protocol's throughput performance to an imaginary protocol that makes optimal routing decisions, based on the network's actual behavior. Under this metric of performance, it is shown in [10] that for *any* (deterministic) protocol, there exists network behavior for which that protocol delivers a factor of n fewer packets than an ideal protocol could have delivered under the same network behavior; in other words, the best achievable "competitive-ratio" is $1/n$ (here, n is the number of nodes in the network). A specific protocol that achieves competitive-ratio of $1/n$ is also presented in [10].

Thus, the greater generality of the network considered in [10] comes at a cost: in practice, a competitive-ratio of $1/n$ is unacceptable. Since $1/n$ is shown to be the best achievable competitive-ratio in the network model of [10], developing protocols with better performance guarantees is only possible if additional assumptions are placed on network behavior.

While the different network model and definition of throughput make direct comparison impossible, many of the techniques used by the protocol in [10] to combat the combined efforts of the edge-scheduling and node-controlling adversaries are extensions of the work presented here.

1.3. *Our Results*

Prior to our work, no protocol has been demonstrated to be provably secure in networks that are simultaneously susceptible to faults occurring due to edge failures *and* faults occurring due to malicious activity of corrupt nodes.[6] 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 is distributed (local control) and 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. 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^5 packets 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, local-control protocol that achieves the following properties in networks susceptible to **any** poly-time conforming Edge-Scheduling and Node-Controlling Adversaries:

- *Correctness.* Achieves correctness with all but negligible (in k) probability of failure

[6] There are numerous protocols that have been designed to work in specific networks settings, e.g. ad hoc wireless networks, where the networks are susceptible to both corrupt/faulty nodes as well as unreliable links. While research in designing specific (end-to-end) protocols for specific network settings is extensive, no existing protocol has been demonstrated to be *provably* secure with respect to a formal notion of security (as presented here).

Table 1. Comparison of our protocol to related existing protocols and folklore. Above, n represents the number of nodes in the network, and P is the size of a packet.

	Secure against edge-sched. ad?	Secure against node-cntr. ad?	Processor memory	Throughput rate x rounds $\rightarrow f(x)$ packets
Protocol of [3]*	YES	NO	$O(n^2P)$	$f(x) = O(x - n^2)$
Protocol of [14]†	YES	NO	$O(nP)$	$f(x) = O(x/n - n^2)$
(Folklore‡) (flooding + signatures)	YES	YES	$O(P)$	$f(x) = O(x/n - n^2)$
(Folklore) (signatures + sequence no.'s)	YES	YES	Unbounded§	$f(x) = O(x/n - n^2)$
Our protocol	YES	YES	$O(n^2P + n^4(k + \log n))$	$f(x) = \Omega(x - n^2)$

*The analysis in [3] is concerned with *memory* costs of its Data Dispersal Algorithm (a variant of the Slide protocol) in *asynchronous* networks. The throughput rate indicated in the table comes from the analysis of the variant of the Slide protocol presented in Sect. 4 of the current paper.

†The analysis in [14] is concerned with *memory* costs of the protocol in *asynchronous* networks. In particular, there is no analysis of the performance of the protocol with respect to *throughput-rate* in a *synchronous* network setting. The throughput rate indicated in the table represents our analysis of the protocol in [14] and is not proven rigorously in this paper or in [14].

‡The usefulness of the flooding technique to combat Byzantine attacks has been investigated by numerous authors, beginning with [16].

§In order for a packet's position to be described in $\Theta(\log n)$ bits (as is assumed in our model), the number of packets in the input stream must be polynomial in n. While the memory required of this folklore protocol can be bounded with respect to the size of the input stream, by "unbounded" we mean that the degree of the polynomial that describes the size of the input stream can be arbitrarily large in n.

- *Throughput.* Achieves *linear* throughput: For any $x \in \Omega(n^5)$, transmits x packets in $O(x)$ rounds
- *Memory.* Requires $O(n^4(k + \log n))$ memory per processor

While the protocol introduced in this paper provides an important first step in establishing the feasibility of throughput-efficient routing in highly unreliable networks, we emphasize that our protocol falls short of providing a practical routing solution for the following reasons. First, the protocol introduced in Sect. 5 is fairly complex, requiring multiple pages of pseudo-code to describe it in detail. Second, our protocol assumes a synchronous network, and it remains an open problem of extending it to asynchronous network settings. Finally, our protocol is expensive in terms of both processor memory and computation. As shown in Table 1 and discussed in Sect. 2, both the complexity and processor costs come at the expense of guaranteeing optimal throughput (e.g. otherwise one could use Flooding + Signatures protocol). As this is primarily a feasibility result, we have put at a premium the requirement of provably correct routing in networks that are susceptible to deliberate, malicious attacks on any subset of the nodes and edges. It is worth noting that in practice, one typically values protocols that require less processor-memory and processor-computation, and enjoy greater simplicity and/or observed throughput; characteristics that can be obtained by relaxing the corruption model and/or removing the requirement of perfect correctness.

2. Challenges and Naïve Solutions

The Slide protocol (and its variants) have been studied in a variety of theoretical contexts, including multi-commodity flow (Awerbuch and Leighton [5]), and in networks controlled by an online bursty adversary (Aiello et al. [4]).

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*[7] 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.[8] 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 received by the receiver.

One idea to try to improve the throughput rate might be to have the sender streamline the process, sending packets with ever-increasing sequence numbers without waiting for n rounds before sending the next packet. In particular, across each of his edges the sender will send *every packet*[9] 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[10]), but additionally this modification requires arbitrarily large[11] processor memory, since

[7] By "flood," we mean that the sender will repeatedly attempt to send the first packet across every adjacent edge for n rounds, and then do the same thing for the second packet for the next n rounds, and so forth. The internal nodes behave similarly, attempting to send whatever packet they are storing (there will be just one) across every adjacent edge. They continue to do this until they have a more recent packet, at which point they repeat this behavior with the new packet (and delete the older packet).

[8] 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.

[9] More precisely, for each neighboring node n_i, the sender keeps track of the highest indexed packet $p_{j(i)}$ it has sent to that neighbor (and gotten confirmation of receipt from the neighbor). Then the next time the sender is able to utilize edge $E(S, n_i)$, it sends the next indexed packet that has not yet been received by the receiver (e.g. packet p_{j+1}, if the sender has not already received confirmation of receipt for this packet from the receiver).

[10] Consider the following pattern of edge failures/activations by the scheduling adversary: for $1 \leq i \leq n-2$ and for any $m \in \mathbb{N}$, during round mi, the only active edges are $E(S, N_i)$ and $E(R, N_i)$. It is straightforward to see that this pattern of activating edges will have this protocol deliver just one *distinct* packet at the end of each $(n-2)$ rounds, for a throughput rate of $\Theta(x/n)$.

[11] Since this paper is concerned only with transferring an (arbitrary) polynomial (in n and k) number of inputs, by "arbitrarily large" processor memory, we mean that there does not exist a memory bound $C = \text{poly}(n, k)$ for which this protocol is correct and achieves reasonable throughput. To see that correctness and/or throughput become problematic if memory is bounded by something polynomial in n (call it C), consider

achieving correctness in the dynamic topology of the graph will force the nodes to remember all of the packets they see until they broadcast them across all adjacent edges or have 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[12] 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 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.

---

the following example. Label the internal nodes $(S, R, N_0, N_1, \ldots, N_{n-3})$, where $n \geq 5$ is the size of the network. Suppose that for $M := \lceil C(1 + 5/n) \rceil$ rounds, the scheduling adversary activates edges as follows: for each $1 \leq i \leq n-3$ and for any $m \in \mathbb{N}$, during round mi, the only active edges are $E(S, N_i)$, $E(N_i, R)$, and $E(S, N_0)$. Notice that at the end of these M rounds, the sender has successfully sent $\approx M/(n-3)$ packets to the receiver (and received confirmation of receipt for these), which means that node N_0 is storing its capacity C packets (the sender has had the opportunity to send M packets to N_0, and at most $M/(n-3)$ can be deleted by N_0 when the sender indicates the packets have already been received by the receiver; so $M - M/(n-3) = C(n+4)(n-4)/n(n-3) > C$). Then for the next $2C$ rounds, the scheduling adversary activates edges $E(S, N_1)$, $E(N_1, N_2)$, and $E(N_2, R)$. Notice that by the end of these $2C$ rounds, $\approx 2C$ packets have gone from sender to receiver through N_1 and N_2, and neither N_1 nor N_2 will be storing any confirmations of receipt for any of the C packets that N_0 is storing. At this point, edges $E(S, N_1)$, $E(N_1, N_0)$, $E(N_0, N_2)$, $E(N_2, R)$ are activated every round henceforth. Notice that no progress can be made, since N_0 cannot accept any more packets until it has freed room, and it cannot delete any of its packets until it receives indication that the receiver already has them (but neither N_1 nor N_2 can give N_0 such confirmation, since they no longer have it). One could try to modify this protocol in various ways (e.g. allow overwriting of packets by packets that have a much higher index number, or have sender/receiver resend old confirmations of receipt); but this example demonstrates the challenge of simultaneously maximizing throughput while demanding correctness, and the fact that naïve protocols do not suffice.

[12] We give a definition of the adversary in Sect. 3.2.

- *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 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 [3], which has enjoyed practical success in networks with dynamic topologies, but is not secure against nodes that are allowed to behave maliciously. We chose the Slide protocol as our starting point because of its proven ability to work well in networks with dynamic topology (with frequent edge failures), see e.g. [2,6], and [3]. Furthermore, the protocol has proven to be robust in its ability to be readily extendible to more complex network settings (which was important for our goal of extending to networks whose nodes cannot be trusted), such as multi-commodity flow [5], networks controlled by an adversary inserting packets in attempt to jam the network [4], and networks with more stringent demands on available processor memory [14].

We provide a detailed description of our version of the Slide protocol in Sect. 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 is to extend the protocol by using digital signatures[13] to provide resilience against Byzantine attacks and arbitrary

[13] 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

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 Sect. 5 and pseudo-code in Appendix C, as well as providing proofs of security in Appendix D. 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. [3]) are secure and efficient against an edge-scheduling adversary. In particular, the following explains how previous authors of the Slide protocol combated the problem of faulty edges in a network. It will be useful to discuss how some of the challenges posed by a network with a dynamic topology are handled. See Sect. 4 for a thorough description of the Slide protocol.

  For the Slide protocol, the total capacity of the stack data-structure is bounded by $4n^3$. That is, each of the n nodes can hold at most $2n$ packets in each of their $2n$ stacks (along each directed edge) at any time.
  - To handle the loss of packets due to an edge going down while transmitting a packet, a node is required to maintain a copy of each packet it transmits along an edge until it receives confirmation from the neighbor of successful receipt.
  - To handle packets becoming stuck in some internal node's stack due to edge failures, *error correction* is utilized. In particular, the sequence of packets $\{p_1, p_2, \ldots\}$ is partitioned into *messages*, with each message containing $\Theta(n^3)$ packets. The messages are then expanded into *codewords*, and then the codewords are divided into *codeword packets* of size P. The transformation from packets to codeword packets has the property that the receiver can "decode" a message (thus obtaining the original $\Theta(n^3)$ packets corresponding to the message), even if it is missing a fraction of the codeword packets. 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

signing "batches" of packets; using private-key initialization [8,12], or using a combination of private-key and public-key operations, such as "on-line/off-line" signatures [11,19]. For the sake of clarity and since the primary focus of our paper is theoretical feasibility, we restrict our attention to the straightforward public-key setting without considering these techniques.

digital signature, and intermediate nodes and the receiver never accept or forward packets 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 Data Dispersal Algorithm (a variant of Slide) of [3], 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 proceeds to begin transferring 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 packet 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.[14] 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 the transmission of a message is not successful, the sender is not only able to *detect* the fact that

[14] 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.

malicious activity has occurred, 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 *testimonies*, which are signatures on the relevant parts of the communication with their neighbors. These testimonies consist of three pieces of information: (1) The net number of packets a node has transferred with each neighbor; (2) For each packet p, the net number of times a node has transferred p with each neighbor; and (3) An integer representing the net "potential drop" a node has had with each neighbor (roughly, this integer relates the relative number of packets each node was storing each time a packet is transferred between them). Note that in terms of memory, the cost of storing these testimonies is $O(n^4)$, controlled by item (2), which has each node storing $O(n^3)$ signatures from each neighbor (there are $O(n^3)$ relevant packets p).

  We prove that no matter what the reason for failure, if the sender has the complete testimony from every node, he 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 (see the "Blacklist" bullet below).
- *Processor Memory.* The signatures on the communication that 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 5.1 in Sect. 5 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 testimonies 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, so that a node need not be connected to the sender for very many rounds in order for the sender to get its information. Specifically, regardless of the reason for failure 2–4 above, a testimony consists of only n pieces of information from each node: 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 do not want honest nodes that are temporarily disconnected from being on this list for too long. We show in Theorem 5.2 and Lemma D.31 that the occasional honest node that gets put on the blacklist will not 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 statement of this fact can be found in Theorem 5.2 in Sect. 5.

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 Sect. 4 we present an efficient protocol ("Slide") that works well in the edge-scheduling adversarial model, and then extend this protocol in Sect. 5 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 assume a synchronous network, so that there is a universal clock that each node has access to.[15] The global time is divided into discrete chunks, called rounds, which consists of two equal intervals of unit time called stages, and all nodes are synchronized in terms of when each stage begins and ends.

We do not assume that the topology of the graph is fixed or known by the nodes. 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 assume a fixed bandwidth/capacity P for each edge; so that an edge that is "up" during a stage can transmit up to P bits of information across it. Our protocol of Sect. 4 requires that $P \in \Omega(\log n)$, while the protocol of Sect. 5 requires that $P \in \Omega(k + \log n)$, where k is the security parameter (discussed in Sect. 3.2 below).

The network is local control, so that the only information that nodes have concerning the state of the network comes from the local communication they have with their neighbors across each edge. During each stage, each node first makes a decision (based on packets it has received in previous stages) about which packets to send across each edge, then it sends these packets, and finally the node receives packets that were sent to it (across edges that were "up" during that stage). In this paper, the constraints we are concerned with in terms of the processors is with respect to processor memory; we ignore computation costs and assume that the computation required of each node at the start of each stage (in terms of making routing decisions) happens instantaneously.

There are two designated vertices, called the sender S and the receiver R, who wish to communicate with each other through this network. We assume the sender has bundled the information he wishes to send the receiver into a sequence of "packets" $\{p_1, p_2, \ldots\}$ of size at most P. 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). As mentioned in the Introduction, the three commodities we care about are Correctness, Throughput, and Processor Memory. We define each of these notions in terms of the network model:

1. *Correctness.* A protocol is correct if the sequence of packets output by the receiver is a prefix of packets that were sent by the sender, without duplication or omission.
2. *Throughput (Rate).* This measures the number of packets that the receiver has obtained 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, i.e. the maximum number of packets and/or control information (measured in bits) an internal node may be required to store at any moment in time. Memory may be a function of the size of the network n, but is independent of the number of packets to be transferred.

[15] 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.

Although the edges in our model are bidirectional, 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 *edges* of the network: at any time, the edge-scheduling adversary controls whether an edge is able to deliver a packet or not. Note that the edge-scheduling adversary only controls the status of the edges, i.e. he cannot duplicate or alter the packets that pass across the edges (aside from preventing them from being delivered by deactivating an edge). Below, we introduce a second adversary that will be able to modify and duplicate packets (as well as many other forms of destructive behavior); we describe a protocol that handles a coordinated attack by *both* adversaries in Sect. 5.

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 edge-scheduling adversary:

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 `t`, we will refer to the path guaranteed by the conforming assumption as the active path of round `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.[16]

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.[17] For a thorough discussion of these notions, see [13] and references therein.

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

[16] The distinction between the two kinds of adversaries is made solely to emphasize the contribution of this paper. Edge-scheduling adversaries (as described above) are commonly used to model edge failures in networks, while the contribution of our paper is in handling the additional presence of a node-controlling adversary, which has the ability to corrupt the *nodes* of the network.

[17] 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.

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 under the control of the node-controlling adversary (although the adversary may choose to have the node behave 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[18] (indeed, any intermediate node is 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.

4. Routing Protocol in the Edge-Scheduling Adversarial Model

In this section we formally describe our edge-scheduling protocol, which is essentially the protocol of [3] (we have modified the specifics in order to fit our network model and allow analysis in this model, but have not changed the original protocol in any substantial way). As mentioned in the Introduction, this protocol is motivated by the Slide protocol developed in [2,6], and [3], and as such we will refer to the protocol presented in this section as "Slide."

4.1. *Definitions and High-Level Ideas*

Recall from Sect. 3.1 that the goal of the protocol is to transmit a sequence of packets $\{p_1, p_2, \dots\}$ of size P from the sender S to the receiver R.

To allow for packets to become stuck in isolated nodes (which may be possible based on the dynamic topology of the network graph, as controlled by the scheduling-adversary), we will utilize *error correction* (see e.g. [13]).[19] Specifically, the packets

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

[19] In this paper, we assume the existence of an error-correcting code with information rate σ and error rate λ.

$\{p_1, p_2, \ldots\}$ are first grouped together into messages $\{m_1, m_2, \ldots\}$, with each message consisting of $\frac{6\sigma n^3}{\lambda}$ packets. The messages are then expanded by a factor of $1/\sigma$ into codewords $\{c_1, c_2, \ldots\}$, and each codeword is partitioned into codeword packets of size P to be transmitted through the network from S to R. We emphasize the distinction between the original *packets* that the sender is ultimately trying to get to the receiver, verses the *codeword packets* that are the actual contents that are relayed through the network. We will frequently use the terminology "packet" when referring to "codeword packets"; relying on context to disambiguate which type of packet we mean.

We assume that the codewords are formed as part of the setup of our protocol. Given the above construction of codewords, we emphasize the following quantity of codeword packets per codeword:

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

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 $(1-\lambda)D = (1-\lambda)(\frac{6n^3}{\lambda})$ packets corresponding to the same codeword, he will be able to decode:

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

Each node will maintain a stack (i.e. FILO buffers) along each of its (directed) edges that can hold up to $2n$ packets concurrently. 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 is not 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 shuffled between the internal buffers of a node between rounds.

Definition 4.1. The height of an incoming/outgoing buffer is the number of packets currently stored in that buffer. Also, the height of a packet refers to its position in the buffer/stack, i.e. one plus the number of packets below it.

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 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.[20] 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.

Codewords are transferred sequentially, so that at any time, the sender is only inserting packets corresponding to a single codeword. We refer to the rounds for which the sender is inserting codeword packets corresponding to the ith codeword as the ith transmission. Lemma 4.9 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 $D - 2n^3$ packets of some codeword, he will clear his outgoing buffers and begin distributing packets corresponding to the next codeword.

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. See Appendix A for pseudo-code.

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

1. *Incoming Buffers.* Recall that we view each bidirectional 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 ('0' for "normal" and '1' for "problem"), 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 4.4 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 can hold up to $2n$ 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

[20] Although the original Slide protocol does not require room to be left in incoming buffers, our technique for proving optimal throughput rate requires this modification: At a high level, the proof regarding throughput will require that a packet never increase in height from the time it is inserted into the network by the sender. Thus, whenever a packet is transferred, the height it assumes in the receiving node's buffer must be no greater than the height it had in the sending node's buffer. While this is the guiding principle of Slide, in our network setting and version of the Slide protocol, this property is threatened in the following scenario: Suppose there is a round in which node A's incoming buffer along $E(A, B)$ has fewer packets than B's outgoing buffer along that edge; a condition that leads the Slide protocol to transfer a packet p_0 from B to A. But as this packet is being exchanged, the connecting edge fails before it is delivered. The edge remains down for some time, during which A's other edges remain active, and A's buffers fill because it is getting packets from other neighbors (our version of Slide keeps its buffers *balanced*, so that even A's incoming buffer along $E(A, B)$ is filling during this time). Then once $E(A, B)$ becomes active again, our protocol must decide what to do with p_0. If no spot was reserved for p_0, then either A rejects p_0, or A accepts p_0 at a height higher than p_0 had in B's buffer (or A must shift other packets *up* to make room for p_0). Both these scenarios are problematic for our throughput proof, the former because our proof requires that an active edge is always utilized when it has a packet to transfer, and the latter because it violates the property that packets always *decrease* in height.

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	$\longleftrightarrow$	$H_B :=$ Height of buffer along $E(A, B)$ Round prev. packet was received
2	Send packet if: • $H_A > H_B \quad OR$ • B did not rec. prev. packet sent	$\longrightarrow$	

Fig. 1. Description of communication exchange along directed edge $E(A, B)$ during the Routing Phase of any round.

"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 packets, which are clustered into messages and expanded into codewords. The codeword packets for the current codeword are distributed to the sender's outgoing buffers whenever there is room for them there.

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

Routing Phase As indicated in Sect. 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 in Appendix A. Informally, Fig. 1 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.

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. 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. Ties for which buffer to choose 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[21] (they will take the first non-occupied slot).

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 the current codeword (Fact 1 says this amount is at most $D - 6n^3$), then R outputs the corresponding message, and deletes all packets corresponding to this codeword from its storage buffer (also, R will not store any packets that it receives in future rounds that correspond to this codeword).

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*.[22] The following theorem

[21] 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 B.13 in Appendix B). 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.

[22] As mentioned, the Slide protocol was developed and analyzed in a series of papers prior to this paper, including [2–6], and [14]. However, none of these papers considered the network setting encountered in the present paper, and as such Theorem 4.3 (and more specifically the fact that the Slide protocol enjoys linear throughput-rate in synchronous networks) has not been proven previously.

concerns the memory requirements of our edge-scheduling protocol, which is bottlenecked by the $O(n^2)$ packets that each internal node has the capacity to store in its buffers.

Theorem 4.2. *The edge-scheduling protocol described in Sect. 4.2 (and in the pseudo-code in Appendix A) requires at most $O(n^2 P)$ bits of memory of the internal processors.*

Proof. Each internal node needs to hold at most $O(n^2)$ packets of size P at any time (nodes have $2(n-2)$ buffers, each able to hold $2n$ packets).[23] □

The throughput standard expressed in Theorem 4.3 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 4.3 is not observed. Note that the theorem below implicitly states that our edge-scheduling protocol is *correct*.

Theorem 4.3. *The transmission of every message m_i takes at most $3D$ rounds; that is, within $3D$ rounds of the time the sender begins inserting packets corresponding to any codeword c_i, the receiver will have received enough codeword packets to decode the message. Thus, for any $x > 3D$, after x rounds R has received $\Omega(x)$ packets from the input stream of packets $\{p_1, p_2, \ldots\}$, and thus our edge-scheduling adversarial protocol enjoys a linear throughput rate.*

The first statement of the theorem implies that for any $y \in \mathbb{N}$, after $3yD$ rounds R will have received at least y messages. This in turn implies the second statement, since each message is comprised of $\Theta(n^3)$ packets from the original packet stream and $D = \frac{6n^3}{\lambda} = \Theta(n^3)$. The proof of the first statement of Theorem 4.3 is rather involved, and will require many lemmas and subclaims that follow from the Routing and Re-Shuffle Rules of Sect. 4.1. We sketch the proof of this theorem below. Pseudo-code, as well as all technical proofs relying heavily on the pseudo-code, can be found in Appendices A and B. We begin with the following definitions:

Definition 4.4. 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.

Definition 4.5. 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 4.4) in round $\mathtt{t}$. Note that this definition does not require that S receives the verification of receipt, so S may not be aware that a packet was inserted.

Definition 4.6. The sender is blocked from inserting any packets in some round $\mathtt{t}$ if the sender is not able to insert any packets in $\mathtt{t}$ (see Definition 4.5). Let $\beta_{\mathtt{T}}$ denote the number of rounds in a transmission $\mathtt{T}$ that the sender was blocked.

[23] Note that for the Slide protocol in this section, $P \in \Omega(\log n)$ to account for the fact that a codeword packet must carry with it $\Omega(\log n)$ bits of information regarding its position in the codeword. In the next section, we will require that $P \in \Omega(k + \log n)$, where k is the security parameter, so that each packet can carry a signature.

The following definition formalizes the notion of "potential," and will be necessary to prove throughput-performance bounds. A good way to think about potential is to imagine each packet contributes to a buffer's potential by an amount proportional to the height of the packet in the buffer. This way, when a packet is transferred from (the top of, since packets are transferred FILO) one buffer to (the top of) another buffer, there will be a drop in overall potential (the sending node will decrease in potential by an amount greater than the increase in potential to the receiving node, based on the routing rules of the Slide protocol). This net potential drop for each packet transfer will be important for arguing a linear throughput rate, see e.g. Lemma 4.11.

Definition 4.7. For any buffer $B \notin S, R$ 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 $N \in G \setminus \{R, S\}$, 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:

$$\Phi_t := \sum_{N \in G \setminus \{R,S\}} \Phi_t^N.$$

It will be useful to break an internal node's potential into two parts. The first part, which we term packet duplication potential, is 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 4.4). 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 decides to send a packet, but the flagged packet's height does not count towards packet duplication potential until the receiving node has accepted the packet (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 the Slide 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 relevant incoming buffer. Meanwhile, the sending node's *non-duplicated potential* drops by the height that the packet occupied in its outgoing buffer, and there is a simultaneous and equivalent *increase* in this sending node's *packet duplication* potential. Note that if we did not introduce the notion of *duplicated potential*, it would not necessarily be the case that network potential never *increases* as a result of a packet transfer. In particular, Lemma 4.11 would no longer be valid, and the proof of Theorem 4.3 becomes more difficult.

Definition 4.8. The height of a packet in an incoming/outgoing buffer is the spot it occupies in that buffer.

We now (restate and) prove the main theorem of Sect. 4.3 (the lemmas required in the proof will be stated within the proof, and proven *after* the proof of the theorem).

Theorem 4.3. *The transmission of every message m_i takes at most $3D$ rounds*; *that is*, *within $3D$ rounds of the time the sender begins inserting packets corresponding to any codeword c_i, the receiver will have received enough codeword packets to decode the message. Thus, for any $x > 3D$, after x rounds R has received $\Omega(x)$ packets from the input stream of packets $\{p_1, p_2, \ldots\}$, and thus our edge-scheduling adversarial protocol enjoys a linear throughput rate.*

Proof. The second statement follows from the first, as discussed above, so it remains to prove the first statement. Let $\mathtt{t}$ denote the round that S first tries to insert packets corresponding to a new codeword b_i into the network. Considering the rounds between $\mathtt{t}$ and $\mathtt{t} + 3D$, we apply the pigeonhole principle to argue that 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 4.9:

Lemma 4.9. *If at any time $D - 2n^3$ distinct packets corresponding to some codeword b_i have been inserted into the network, then R can decode message m_i.*

It remains to prove the theorem in the latter case. Note 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 $2nD$) by Lemma 4.10:

Lemma 4.10. *If at any point in any transmission $\mathtt{T}$, 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 $n\beta_{\mathtt{T}}$.*

Meanwhile, the *increase* to network potential between $\mathtt{t}$ and $\mathtt{t} + 3D$ caused by *duplicated potential* is at most by $2n^3 - 8n^2 + 8n$ by Lemma 4.11:

Lemma 4.11. *Every change in network potential comes from one of the following three 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* $2n^3 - 8n^2 + 8n$.

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.

Note that the potential already in the network at the start of $\mathtt{t}$ adds to the potential at most $4n^4 - 14n^3 + 8n^2 + 8n$:

Claim. *The maximum amount of potential* (*see Definition* 4.7) *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. □

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 4.11 (stated above) also indicates 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 $\mathtt{t}$ and $\mathtt{t} + 3D$, and again R can decode by Lemma 4.9 (stated above). □

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

Lemma 4.9. *If at any time* $D - 2n^3$ *distinct packets corresponding to some codeword* b_i *have been inserted into the network, then* R *can decode message* m_i.

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

Proof. Our protocol dictates that when a node sends a packet to a neighboring node, it maintains a copy of the packet until it gets confirmation of receipt from the neighbor. We restate this as Claim B.14 in Appendix B, where it is proven in terms of the pseudo-code. □

The maximum number of packets that can be stored in some incoming/outgoing buffer of an internal node is bounded by $4n^3$: Each node has $(n-2)$ outgoing buffers (one to each node except itself and S) and $(n-2)$ incoming buffers (one from each node except itself and R), and thus a total of $2(n-2)$ buffers. Each of these buffers has capacity $2n$, and there are $n-2$ internal nodes, so the internal buffer capacity of the network is $4n(n-2)^2$. Therefore, if $D - 2n^3$ distinct packets corresponding to b_i have been inserted, then R has necessarily received $D - 6n^3 = (1-\lambda)(\frac{6n^3}{\lambda})$ of these, and so R can decode message m_i by Fact 1. □

The following Lemma will be useful in bounding the number of rounds in which no packets are inserted.

Lemma 4.10. *If at any point in any transmission* $\mathtt{T}$*, 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* $n\beta_{\mathtt{T}}$.

Proof. The idea 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 guaranteed *active path* for that round) will necessarily have a full incoming buffer along its edge to the sender. The following claim states that buffers are *balanced* at all times, and hence all of N's outgoing buffers are also full:

Claim. *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. The Re-Shuffle rules dictate that if there is ever a buffer whose height is at least two bigger than another buffer, then a packet will be shuffled from the higher buffer to the lower one. Similarly, packets are preferentially taken from incoming buffers and shuffled to outgoing buffers if there is ever an incoming buffer with larger height than an outgoing buffer. We restate this as Claim B.3 in Appendix B, where it is proven in terms of the pseudo-code. □

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, near the front-end 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,[24] 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.

This lemma is restated in Lemma B.18 in Appendix B, where we prove it based on the pseudo-code of our protocol. □

Lemma 4.11. *Every change in network potential comes from one of the following three 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. Note that all packet movement falls under one of two categories: (1) Packets transferred between two nodes; and (2) Packets re-shuffled between the buffers on one node. Both of these fall 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, 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 4.7). Also, since only flagged packets (but not necessarily all of them) contribute to network packet duplication potential, the largest this value can have is the maximum height of a flagged packet times the maximal possible number of flagged packets in the network. By the fact that outgoing buffers have at most one flagged packet at any time,[25] there are at most $(n-2)^2$ flagged packets in the network at any given time, and each one has maximal height $2n$ (the maximum capacity of each buffer), so network packet duplication potential is bounded by $2n^3 - 8n^2 + 8n$.

[24] 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.

[25] Intuitively, an outgoing buffer has at most one flagged packet since the buffer will continue trying to send this packet until it receives confirmation of receipt from the neighboring node. We restate and prove this fact in terms of the pseudo-code in Claim B.10 in Appendix B.

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 places where there is packet movement in our protocol, and argue each will result in a non-positive change to non-duplicated potential. Clearly potential changes caused by *re-shuffling* packets is non-positive, since the re-shuffle rules dictate that packets will only be re-shuffled if they decrease in height or stay the same height in the new buffer.

Meanwhile, when a packet is sent across an edge between two nodes, there are two possibilities: the receiving node *accepts* the packet (as in Definition 4.4), or the receiving node has already accepted this packet (but the sending node is sending it again because it never got confirmation of receipt). In the latter case, the packet is not stored by the receiving node, and the sending node's copy contributes towards *duplicated* potential within the outgoing buffer, so non-duplicated potential is not affected. In the former case, the flagged packet in the sending node's outgoing buffer still counts towards non-duplication potential. 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, 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, which follows from the protocol rules which specify a packet transfer should occur if and only if the sending buffer has height strictly larger than the receiving buffer.[26]

This lemma is restated in Lemma B.4 in Appendix B, where we prove it based on the pseudo-code of our protocol. □

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

In this section we introduce a routing protocol for networks susceptible to *both* edge failures and corruptible nodes. The protocol will be an extension of the Slide protocol presented in Sect. 4, with added mechanisms to handle the fact that nodes cannot be relied upon to behave honestly (i.e. they may deviate from protocol specifications). We will refer to this protocol as "Mal-Slide," to emphasize it is (as will be proven below) secure against a *malicious* node-controlling adversary.

The main difference between the Slide protocol of Sect. 4 and the Mal-Slide protocol will be the introduction of control information, which will contain the relevant pieces of (signed) information necessary to identify a corrupt node. Notice that the *conforming* assumption placed on the node-controlling adversary implicitly states that the sender and receiver are incorruptible. The Mal-Slide protocol makes use of this by dictating that these two nodes (and primarily the sender) bear the burden of detecting *faults*

[26] Even if the sending and/or receiving node's buffer heights have changed since the original communication that prompted the packet transfer, the fact that flagged packets are not re-shuffled or shifted down to fill-gaps ensures that the flagged-packet maintains its original height; while the use of ghost packets ensures that the height assumed by the transferred packet (when it is finally accepted) is exactly one greater than the originally communicated height.

(transmission failures caused by misbehaving nodes), and localizing the faults to the offending node(s). In particular, the control information used to identify a corrupt node will be collected by the sender, and we must allocate system resources (e.g. processor memory and bandwidth) to store this information and transfer it through the network to the sender.

Intuitively, the Mal-Slide protocol can be thought of as cycling through two disjoint phases:

Routing Phase. Codeword packets are transferred through the network from the sender to the receiver, using the same protocol rules as in the Slide protocol of Sect. 4. The only two differences will be: (1) All codeword packets are signed by the sender upon insertion (for authentication, i.e. to protect against packet modification and/or the insertion of junk packets), and this signature accompanies the codeword packet until it is received by the receiver; and (2) One packet of "control information" is piggy-backed onto the transfer of codeword packets, in a manner to be described below.

Regulation Phase. When a message transmission fails (the receiver could not decode the current codeword after the Routing Phase), the Regulation Phase begins. The internal nodes send back the control information to the sender, who uses it to identify a corrupt node. Once identified, corrupt nodes are eliminated from the network, meaning all nodes are forbidden to communicate in any way with corrupt nodes.

In actuality, these two phases happen concurrently, so that the Mal-Slide protocol does not waste time waiting for the control information to make its way back to the sender before beginning the next Routing Phase. Instead, the control information will be transmitted to the sender by piggy-backing it onto the ordinary communication of each round, in a manner to be described below.

In the next four subsections, we describe more precisely the Mal-Slide protocol, and in particular exactly what control information is collected, how it is transferred back through the network to the sender, and how this information is used by the sender to identify corrupt nodes. We then state and prove in Sect. 5.5 theorems concerning the *correctness*, *throughput*, and *memory requirements* of the Mal-Slide protocol.

5.1. *Control Information*

Anytime two nodes transfer a codeword packet between them, they will also transfer (signed) control information that contains values the nodes have been storing and updating through the course of the transmission. In particular, for any pair of nodes (A, B) and directed edge $E(A, B)$:

1. Nodes A and B maintain a running tally of the total number of codeword packets A has sent to B in the current transmission.
2. Every time A sends a packet to B, the packet had some height h in A's outgoing buffer, and the packet assumed some height h' in B's incoming buffer. The height difference $h - h'$ represents the *potential drop* that resulted due to the packet transfer (see Definition 4.7), which was shown in Lemma 4.11 to be always non-negative (assuming honest behaving nodes). Nodes A and B maintain a

running tally of the cumulative potential drop as a result of codeword packets A has sent to B in the current transmission.

3. For every codeword packet p that A has sent to B in the current transmission, A and B maintain a running tally of the total number of times A has passed this specific packet to B.

Node A sends the updated value of each of these three quantities, together with a timestamp indicating the current round and transmission and his signature on all of these items, during Stage Two communication with B (see Fig. 2 below). Note that the values that are sent by A reflect the most recent values of the three quantities in (1)–(3), i.e. the changes made to these quantities based on the current packet being transferred have been updated in the values A sends to B.

Although the routing of codeword packets in the Mal-Slide protocol is the same as the basic Slide protocol, because corrupt nodes are not guaranteed to behave honestly in the present network model, Theorem 4.3 (which guaranteed the receiver could decode the message after $3D$ rounds) is no longer valid. In particular, after the $3D$ rounds of a transmission in the Mal-Slide protocol, the receiver might not have received enough (valid) codeword packets to decode the message. We will refer to such transmissions as failed transmissions. Below we outline three types of malicious behavior that corrupt nodes may engage in to force a transmission to fail, and how the control information can be used to identify a corrupt node in each case. Although this list is in no way intended to be comprehensive of all possible forms of malicious behavior, it turns out that the mechanisms we put in place to handle the following three issues will be enough to handle *all* forms of malicious behavior (not just the ones listed below).

1. *Packet Deletion.* Suppose that corrupt nodes refuse to forward on the packets they receive, so that by the time the sender has inserted all D of the codeword packets, the receiver has not received enough of them to decode the message.[27] In this case, there is necessarily a corrupt node that is deleting packets (or equivalently, storing more packets than it is allowed to). The sender can identify the corrupt node if he can find a node who *input* x packets and *output* y packets, where $x - y > 4n(n-1)$ (the quantity on the right-hand side of the inequality represents the total capacity of an internal node to store packets: it has $n-1$ incoming and outgoing buffers, each of capacity $2n$). Control information of type (1) provides exactly this information (provided the sender can collect this information from all nodes).
2. *Packet Duplication.* Suppose corrupt nodes are duplicating packets in such a manner that keeps nodes "near" the sender (assume for the moment a relatively stable network topology) at full capacity, thus making it impossible for the sender to insert packets (even along the active honest path; recall Definition 3.2). In particular, suppose this strategy prevents the sender from inserting all D codeword packets by the time $3D$ rounds have passed. This means that there have been at

[27] Note that our use of error correction allows for the fact that some packets may be in the internal nodes' buffers, and yet the receiver can still decode: Even if every buffer of each internal node is completely full, the receiver will still be able to decode the codeword if it has received the rest of the codeword packets. Therefore, if the sender has inserted all D of the codeword packets, the receiver should be able to decode, even if many of the packets are stuck in the internal nodes' buffers.

least $2D$ rounds in which the sender was blocked from inserting any packets (see Definition 4.6). Note that Lemma 4.10 of Sect. 4.3 concerns only nodes along the active honest path, and in particular, since these nodes are *honest*, the lemma remains valid (it is restated and proven in Lemma D.14 in Appendix D for completeness). This lemma states that these (at least) $2D$ blocked rounds will cause a recorded potential drop of at least $2nD$, and this drop is recorded in the control information of type (2). Since the sender inserted fewer than D packets in this case, the cumulative recorded *increase* in potential as a result of these insertions is less than $2nD$ (a single packet insertion can cause potential to raise by at most $2n$, since this is the highest height a packet may be stored in an incoming buffer). Since the overall *decrease* in potential (at least $2nD$) outweighs the overall *increase* (less than $2nD$), there is necessarily a node that is responsible for more potential *loss* than *gain*. With control information of type (2) collected from all of the nodes, the sender can identify such a node, which is necessarily corrupt.

3. *Packet Deletion + Packet Duplication.* Suppose that the adversary recognizes he will be caught if he only employs a strategy of packet deletion or packet detection (as in Cases (1)–(2) above). Instead, the adversary forces corrupt nodes to replace valid packets they receive with old packets they have already sent forward. This way, their actions appear consistent in terms of analyzing the control information of types (1) and (2). Notice that transmissions when this occurs can now fail, despite the fact the sender was able to insert all D packets of the codeword *and* the receiver got $(1-\lambda)D$ packets (which is ordinarily what is needed to decode): If too many of the received packets are duplications, then the receiver may not have the $(1-\lambda)D$ *distinct* packets required for decoding. In this case there is at least one packet p that the receiver has received more than once. Since packets are never duplicated by honest nodes (they will never resend a packet before getting an acknowledgement of receipt from the node they sent the previous copy to), the sender can identify a corrupt node by finding a node that *output* the duplicated packet p more times than that node *input* p. Notice that the sender can identify such a node with control information of type (3) (provided the sender can collect this information from all nodes).

The following four cases clearly cover all possible outcomes for a transmission. Notice that the first case corresponds to a *successful* transmission, while the latter three are *failed* transmissions, and they roughly correspond to the three above malicious strategies (F2 corresponds to Packet Duplication, F3 to Packet Deletion, and F4 to Packet Deletion + Duplication).

S1. The receiver was able to decode the codeword within $3D$ rounds
F2. The receiver could not decode, and the sender inserted less than D packets in $3D$ rounds
F3. The receiver could not decode, the sender inserted D packets, and the receiver did *not* receive any duplicated codeword packets
F4. The receiver could not decode and cases F2 and F3 do not happen. In other words, the sender inserted D packets, and the receiver could not decode because he received at least one duplicated packet p.

Notice that there are two forms of control information: (1) The *current* information, which consists of the three quantities mentioned in the section above, and which are being updated and signed for every new packet transfer between two neighbors; and (2) Control information pertaining to *previous* (failed) transmissions, which represent the values that each of the three types of quantities had at the *end* of the earlier failed transmission in question. We emphasize the difference between these two forms of control information: the former kind is being stored and continuously updated between every pair of nodes, and no attempt is made to transmit this information beyond the two nodes it pertains to. If a transmission fails, then the values that each of the three quantities had at the end of that transmission are locked (i.e. they will not be updated/changed in the next transmission), and now (honest) nodes will attempt to transmit the final values of these quantities through the network back to the sender, where they will be collected and used to identify a corrupt node. To distinguish between these two different forms of control information, we will refer to the information for the *current* transmission as control packets and the information that corresponds to *previous* failed transmissions as a node's testimony for the failed transmission in question.

Notice that a node's testimony consists of $n - 1$ packets, i.e. one packet for each neighbor. In particular, for a given node A and one of its neighbors B, the node will return in its (signed) testimony the final (value, signature) pairs on the relevant three quantities from the control packets.[28]

In the next section, we describe exactly the protocol rules for how the testimonies are communicated through the network. We state once-and-for-all that if a node ever receives a packet from a neighbor that has faulty information (e.g. a control packet that does not reflect accurate values or has a faulty signature, or a codeword packet that does not carry the sender's signature), then the node ignores all communication from the offending neighbor for that round (treating the edge as having failed for the entirety of the round).[29]

[28] In the case a transmission $\mathtt{T}$ fails as in F4, which means the receiver got at least one duplicated packet p, then the testimonies from the nodes includes only the final value (from each of their neighbors) on the number of times p has passed between them. In particular, even though the control packets passed between neighbors have been accumulating information on *all* of the $\Theta(n^3)$ codeword packets, the only value returned to the sender is on the relevant packet p that had been duplicated. This way, a testimony from each node consists of $n - 1$ packets instead of $\Theta(n^3)$ packets. Indeed, once a node learns a transmission fails for reason F4 and they know a packet p that has been duplicated (how this knowledge is conveyed is discussed below), they can delete the information corresponding to all of the other $\Theta(n^3)$ packets, freeing up this memory.

[29] An honest node will never send a packet with faulty information. Therefore, if a node receives a packet with faulty information from a neighbor, the node can be certain its neighbor is corrupt. There are three reasonable candidates for how a protocol should have nodes deal with the knowledge its neighbor is corrupt: (1) Do nothing; (2) Keep the information local, but refuse all future communication with the node; (3) Report the offending node, to alert the network the node is faulty in attempt to have it eliminated. Option (3) introduces the difficulty of a "he said, she said" problem of the sender not being able to pinpoint which of the two nodes is corrupt, and we therefore do not use that approach. Of the other two options, we choose the former, simply because it is more in line with our approach for eliminating corrupt *nodes*, as opposed to eliminating the *links* adjacent to corrupt nodes. However, employing strategy (2) can be done without any major modifications to Mal-Slide (although this strategy will not improve the bound for throughput efficiency that is stated in Theorem 5.2 below).

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 Ack. of rec. of Control Info.	$\longleftrightarrow$	$H_B :=$Height of buffer along $E(A,B)$ Round prev. packet was received Control packet for edge $E(A,B)$ Ack. of rec. of Control Info. (Values pertain to p. rec'd in *prev.* round)
2	Send packet and control packet if: • A has rec'd *SOT*, AND • $A, B \notin$ blacklist or eliminated, AND – $H_A > H_B$ OR – B did not rec. prev. packet sent	$\longleftrightarrow$	Receive packet if: • B has rec'd *SOT*, AND • $A, B \notin$ blacklist or eliminated Send Control Info: (1) *EOT*, (2) *SOT*, (3) Nodes to *remove* from BL, (4) Testimonies

Fig. 2. Communication exchange along directed edge $E(A, B)$ during some round.

5.2. *Gathering Control Information*

As mentioned at the beginning of Sect. 5, intuitively the Mal-Slide protocol can be viewed as cycling between a *routing* phase and a *regulation* phase. However, if these two phases are separated in practice, a problem is encountered: How long should the sender wait during the regulation phase to gather the testimonies he requires to identify a corrupt node? If the sender waits for *all* the information he needs before returning to transmitting the next message, then a set of corrupt nodes can refuse to return their testimonies. Since the sender cannot see the status of the edges of the network, he is unable to determine if these nodes are "playing dead" (refusing to use the links available to them to transmit their testimonies) and thus should be eliminated as corrupt, or if they are honest nodes that are (temporarily) disconnected from the rest of the network. Even though the sender will necessarily be getting *some* control information back (e.g. from the honest nodes that are "near" him on the active honest path), this information may not be enough to identify a corrupt node. Furthermore, the sender cannot simply eliminate non-responding nodes from the network, as it is possible that they are honest, and that at some point in the future they will form a crucial link on the active honest path of later rounds.

The Mal-Slide protocol avoids the problem of an indefinite regulation phase by performing regulation tasks in conjunction with the actual routing. Figure 2 describes how the routing and regulation phases are combined.

Notice that *control packets* (control information corresponding to the *current* transmission) are sent in both directions in each round: B sends one to A during Stage 1, and A sends one to B in Stage 2 (the values B uses for each of the three relevant quantities in the control packet it sends to A are current as of the *previous* packet that B received from A, whereas the values A uses in the control packet it sends to B are current as of the *current* packet being sent). A relevant *testimony packet* that B has (either his own or that of another node) can be sent from B to A in Stage 2. As can be seen from Fig. 2, B may potentially have multiple packets of control information (e.g. testimonies) that he would like to send to A during Stage 2. Since B can only send one packet of control information during Stage 2, we describe the rules for how B determines which packet

to send below (after describing the remaining forms of control information found in Fig. 2).

There remains one last change between the Slide protocol of Sect. 4 and the Mal-Slide protocol: the identification of corrupt nodes requires that the sender has collected enough control information from the internal nodes. The procedure for how the Mal-Slide protocol handles the collection of this information, and in particular how it prevents a (set of) corrupt node(s) from delaying the collection of this data by refusing to return information that will implicate themselves, is the content of the next section.

5.3. *The Blacklist*

As discussed above, when a transmission fails as in Case F2, F3, or F4, the sender will request nodes to return testimonies so that the sender can identify a corrupt node. Loosely speaking, the sender maintains a blacklist consisting of all nodes for which the sender has not yet collected their complete testimony (recall that each node's testimony consists of $n-1$ packets, corresponding to the three quantities recorded from the node's communication with its $n-1$ neighbors). Blacklisted nodes are not allowed to transfer any *codeword* packets.

More precisely, when a transmission $\mathtt{T}$ *fails*, *all* nodes who participated in the transmission (i.e. nodes that were *not* already on the blacklist for at least one round during $\mathtt{T}$) are added to the blacklist, and the sender records $\mathtt{T}$ as the transmission these nodes were added to blacklist. A node is not removed from the blacklist until the sender receives the node's complete testimony. Meanwhile, nodes on the blacklist are not allowed to send or receive any codeword packets.[30]

Notice that each failed transmission has a set of nodes that were blacklisted at the end of that particular transmission: the nodes who were not already on a previous blacklist. However, nodes are never on more than one transmission's blacklist, as if they were blacklisted for the entirety of transmission $\mathtt{T}$ that fails, then they are not added to the blacklist for $\mathtt{T}$. After all, the blacklist serves two purposes: (1) It ensures that corrupt nodes cannot continue to cause transmissions to fail while withholding their testimonies from the sender; and (2) It provides a list of nodes that *participated* in each failed transmission. As will be shown in the next section, the sender will be able to identify a corrupt node as soon as there exists a failed transmission $\mathtt{T}$ in which the sender is able to collect the complete testimony from all nodes that participated in the transmission; i.e. nodes that were *not* on the blacklist during at least one round of $\mathtt{T}$. Because blacklisted nodes are not supposed to be transferring codeword packets, if a transmission $\mathtt{T}$ fails, then the sender does not need the testimony from any node that was already blacklisted

[30] Corrupt nodes cannot be prevented from transferring packets to/from blacklisted nodes. However, such a strategy will not help them to cause transmissions to fail without their guilt being discovered, as if they employ this strategy to force a transmission $\mathtt{T}'$ to fail, then either: (1) They return the accurate control information for $\mathtt{T}'$ (in which case the sender sees they transferred codeword packets with blacklisted nodes and are therefore necessarily corrupt); (2) They return the control information to the sender, but do not include with this the parts that indicate they transferred packets with blacklisted nodes (in which case there will be discrepancies in the testimony that they return); (3) They do not return any control information, and are consequently blacklisted (and not able to affect future transmissions) until they return enough control information to fall under case (1) or (2) and become permanently eliminated.

for the entirety of $\mathtt{T}$ (all values in these testimonies should be zero, since the nodes were not allowed to transfer any codeword packets).

After a failed transmission, *every* node (except for the sender, but including the receiver) is therefore on the blacklist (they were either added at the end of the transmission that just failed, or they were already on the blacklist for an earlier failed transmission). Note that even though nodes on the blacklist are not allowed to transfer *codeword* packets, they *are* allowed to transfer control information (see Fig. 2).

In the following section we collect the ideas from Sects. 5.1–5.3 and describe how each transmission $\mathtt{T}$ progresses.

5.4. *Overview of the Mal-Slide Protocol*

The following steps describe the procedure for every transmission of the Mal-Slide protocol (pseudo-code is given in Appendix C).

1. The sender begins each transmission by forming the Start Of Transmission (SOT) broadcast. This consists of:
   (a) A single packet indicating how many total packets will comprise the SOT for the current transmission $\mathtt{T}$ (i.e. how many eliminated and blacklisted nodes there are)
   (b) The list of eliminated nodes
   (c) The list of blacklisted nodes; this includes for each blacklisted node the transmission in which the node was placed on the blacklist

   As mentioned above, if the previous transmission $\mathtt{T}-1$ *failed* (as in Cases F2–F4 above), then the sender blacklists all nodes that were not already on the blacklist, and indicates these new nodes were blacklisted due to transmission $\mathtt{T}-1$. Additionally, for nodes that were blacklisted as a result of a transmission that failed as in F4, the sender indicates the packet p for which the receiver got at least two copies in that failed transmission (the sender has access to this information from the End Of Transmission packet, see Item (5) below). Notice that the SOT broadcast consists of at most n packets, as every node (other than the sender) is either eliminated or on at most one blacklist.

   Each of the above three items are timestamped with the current transmission index $\mathtt{T}$ and signed by the sender. Notice from Fig. 2 that SOT packets are sent during Stage Two communication.
2. Nodes are not allowed to transfer codeword packets until they have received the SOT for the current transmission. This way, no node is (legally) transferring any packets from the current codeword until they have an updated view of the blacklist and eliminated nodes. When the sender has sent the complete SOT to a neighbor, he may begin inserting codeword packets to that neighbor (assuming the neighbor is not blacklisted).
3. If a node ever learns it has been blacklisted (from the SOT broadcast), it will form its testimony: the final values of the three types of control information from each of its neighbors from the indicated transmission. The $n-1$ packets of the testimony are then queued for delivery to the sender. The mechanism for transmitting testimonies (and indeed all forms of control information sent in Stage Two, see Fig. 2) back to the sender is *flooding*: a node will send *every* testimony packet it

has seen (its own testimonies and the testimonies it has collected from its neighbors) across *every* adjacent edge. We note that because a node is on at most one blacklist at any time, and because the testimony of a single node consists of $n-1$ packets, a node need store and transfer at most n^2 testimony packets at any time. Also, the priority of sending control information in Stage Two (see Fig. 2) ensures that nodes know the most recent transmission that each blacklisted neighbor was placed on the blacklist before touching any of its neighbors' testimonies (this way a node knows which of its neighbors testimony packets are current and valid).

We note that sending control information back to the sender via *flooding* does not affect the overall throughput efficiency of Mal-Slide because there is much less control information than codeword packets: $O(n^2)$ verses $\Theta(n^3)$, respectively. In particular, a protocol that employs flooding for *codeword* packets sent from sender to receiver may suffer in a factor of $1/n$ in terms of throughput efficiency. However, because there are only n^2 packets of control information that need to be transmitted to the sender in any transmission, the fact that transmissions last $\Theta(n^3)$ rounds means that the even a loss of $1/n$ in throughput efficiency for these packets will not impede their ability to reach the sender by the end of the transmission.

4. When the sender receives the complete testimony of a blacklisted node, it removes the node from the blacklist. In particular, the sender creates a (signed and timestamped with the index of the current transmission number) packet indicating the node to be cleared from the blacklist, and queues this packet for delivery with the rest of the control information sent during Stage Two (see Fig. 2). If the sender ever has received enough of the testimonies to eliminate a corrupt node, then he starts the current transmission over again, including in the new SOT broadcast the identity of the node that has just been eliminated.
5. If at any time:
   (a) The receiver can decode the current codeword, OR
   (b) The receiver has received a duplicated packet p

   then the receiver forms the End Of Transmission (EOT) packet. This packet contains the label of the duplicated packet p (in Case (5b)), or else a bit indicating successful decoding, and is signed and timestamped by the receiver and queued for delivery with the rest of the control information (see Fig. 2). The EOT packet is used to alert the sender to end the current transmission. We restate now (using current terminology) the four ways a transmission can end. Notice that one of them necessarily happens within $3D$ rounds of starting a transmission, and hence every transmission (whether successful or failed) lasts at most $3D$ rounds:
   S1. Sender receives EOT packet indicating Receiver was able to decode current codeword
   F2. $3D$ rounds have passed, and Sender has not been able to insert D packets, nor has he received an EOT packet
   F3. Sender has inserted D packets but has *not* received an EOT packet
   F4. Sender receives an EOT packet indicating Receiver got some packet p twice

5.5. *Analysis of the Mal-Slide Protocol*

In this section we present the main ideas for why the Mal-Slide protocol is secure against the combined attack of the edge-scheduling and node-controlling adversaries, and analyze the performance of the Mal-Slide protocol in terms of correctness, memory, and throughput. Pseudo-code for Mal-Slide can be found in Appendix C, and proofs of all of the below lemmas and theorems are proven in terms of the pseudo-code in Appendix D.

Theorem 5.1. *The Mal-Slide protocol requires at most* $O(n^2P + n^4(k + \log n))$ *bits of memory of the internal nodes.*

Proof. The most memory-intensive cost of the Mal-Slide protocol is the requirement for nodes to store control information of type (3): For every codeword packet p, each node must store the total number of times it has transferred p to each neighbor. There are $\Theta(n^3)$ packets per codeword, each node has $O(n)$ neighbors, the packet's position within the codeword and timestamp[31] can be described using $\Theta(\log n)$ bits, and it costs $\Theta(k)$ bits to store signatures on each of these, so the overall cost of storing this information is $O(n^4(k + \log n))$. Note the other information a node needs to store (current codeword packets and all other kinds of control information) collectively only require $O(n^2P)$ bits of memory, where $P \in \Omega(k + \log n)$. □

The following theorem, which states that the Mal-Slide protocol has the same (asymptotic) throughput rate as the basic Slide protocol of Sect. 4, implicitly guarantees that the Mal-Slide protocol is *correct*.

Theorem 5.2. *Except for at most* $n^2/2$ *transmissions that may fail due to malicious activity, the Mal-Slide protocol enjoys linear throughput. More precisely, after* x *transmissions, the receiver has correctly decoded at least* $x - n^2/2$ *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* $\Theta(n^3)$ *packages from the original stream of packets, information is transferred through the network at a linear rate.*

Proof. Having the sender sign all packets ensures that the final message that the receiver decodes in each of these transmissions is unaltered (ensuring correctness). We showed at the end of the previous section that every transmission lasts at most $3D = \Theta(n^3)$ rounds. Therefore, transmissions that are *successful* (Case S1) enjoy a linear throughput rate: $\Theta(n^3)$ packets from the original input stream have been received in $O(n^3)$ rounds. It remains to show that there are at most $n^2/2$ failed transmissions, which follows from the following two lemmas (proven after this proof):

Lemma 5.3. *There can be at most* $n - 1$ *failed transmissions before the sender necessarily has all testimonies corresponding to one of these failed transmissions.*

[31] Because our protocol is only valid for an input stream of packets of size polynomial in n, timestamps can be achieved using $\Theta(\log n)$ bits.

Lemma 5.4. *If there is a transmission* T *for which the sender has collected the complete testimonies from every node that participated in* T*, then the sender can necessarily identify a corrupt node.*

Lemma 5.3 guarantees that there can be at most $n-1$ failed transmissions before the sender has necessarily collected the complete testimony of every node who participated in one of these transmissions, and then the sender will be able to eliminate a node by Lemma 5.4. This effectively reduces the network to $n-1$ nodes, and we can repeat this argument to ensure that there can be a total of at most $n^2/2$ failed transmissions. □

Lemma 5.3. *There can be at most $n-1$ failed transmissions before the sender necessarily has all testimonies corresponding to one of these failed transmissions.*

Proof. (Sketch) As discussed in Sect. 5.3, a node can be on at most one blacklist at any time. In particular, if a node participated in some failed transmission T, then the node is not allowed to participate in any future transmission until the sender receives the node's complete testimony. We begin with the following observation:

Observation. *The receiver* participates *in every transmission, i.e. by the end of every transmission, the sender has received the receiver's complete testimony (if the receiver was on the blacklist*[32] *for the previous transmission).*

Proof. (Sketch) This follows from the *conforming* assumption (which guarantees that every round has an *active* path going through *honest* nodes) together with how testimonies are relayed through the network to the sender. In particular, in any transmission there are at most $\Theta(n^2)$ packets of control information (includes SOT, EOT, nodes to remove from the blacklist, and testimony packets). Because the receiver is linked to the sender in each of the $3D$ rounds of the transmission, its information will take at most $\Theta(n^3)$ rounds to reach the sender, where the constant in the Θ is small enough to ensure the sender has the receiver's information by the time $3D$ rounds have passed. This observation is restated and proven in terms of the pseudo-code in Lemma D.8 of Appendix D. □

Therefore, if there have been $n-2$ failed transmissions and the sender has not collected all of the testimonies from participating nodes for any of the transmissions, then in all subsequent transmissions, either the sender completes his knowledge of the testimonies for a failed transmission, or just the sender and the receiver participate in the transmission. In the latter case, the transmission is guaranteed to be *successful*, since both the sender and receiver are honest, and the active honest path must have been a direct link between them for the vast majority of this transmission (otherwise, if there is some honest node A that is part of numerous active honest paths during the transmission,

[32] Even though the receiver cannot be corrupted, he nevertheless can (and will) be placed on the blacklist at the end of each failed transmission. After all, the blacklist is not just a list of potential nodes that may be corrupt; rather, it is a list of nodes who participated in a given failed transmission and it emphasizes that the sender has yet to receive those nodes' testimonies for the transmission. So the receiver gets placed on the blacklist to emphasize to intermediate nodes the need to relay the receiver's testimony back to the sender.

then the sender will necessarily receive all of A's outstanding testimony, completing his knowledge of all testimonies for some failed transmission). This lemma is restated and proven in terms of the pseudo-code in Lemma D.9 of Appendix D. □

Lemma 5.4. *If there is a transmission* $\mathtt{T}$ *for which the sender has collected the complete testimonies from every node that participated in* $\mathtt{T}$*, then the sender can necessarily identify a corrupt node.*

Proof. (Sketch) Failed transmissions fall under Case F2, F3, or F4. The mechanism for how the testimonies can be used to identify a corrupt node in each of these cases was described in Sect. 5.1 above. Below we go into a little more detail, but reserve the formal proof with references to pseudo-code in Lemma D.27 in Appendix D.

Case F2. In this case, the Sender has the complete testimony from every participating node for a transmission that failed due to Case F2. We will use control information of type 2 to identify a corrupt node. The key observation is to note that a variant of Lemma 4.10 from Sect. 4 remains valid when applied to the active honest path that exists each round, since all nodes on this path are honest (this is proven in Lemma D.14):

$$-n \geq \sum_{N \in \mathsf{P}_{\mathtt{t}}} \Delta\Phi_N,$$

where $\mathtt{t} \in \beta$ is any blocked round, and $\mathsf{P}_{\mathtt{t}}$ denotes the set of nodes comprising the active honest path for round $\mathtt{t}$ (as guaranteed by the *conforming* assumption). Since the Sender was unable to insert D packets (otherwise we would be in Case F3), the number of blocked rounds is at least $2D$ (transmissions that fail as in Case F2 have $3D$ rounds), and hence:

$$-2nD \geq \sum_{\mathtt{t} \in \beta_{\mathtt{t}}} \left(\sum_{N \in \mathsf{P}_{\mathtt{t}}} \Delta\Phi_N \right). \tag{2}$$

Meanwhile, potential only *increases* on packets inserted by S; intuitively this is Lemma 4.11, which is proven for the malicious model in Lemma D.11. Since S inserted fewer than D packets (we are in Case F2), and each inserted packet adds at most $2n$ to total potential (the maximum height a packet can have in a buffer is the capacity of the buffer), we have that the total increase in potential due to packet insertions in the failed transmission in question obeys:

$$2nD > \Delta\Phi^*, \tag{3}$$

where $\Delta\Phi^*$ represents the changes in potential caused by all packet insertions. Therefore, we have that the total change in potential during this failed transmission satisfies

$$\begin{aligned} \Delta\Phi = &(\text{Cumm. inc. from packet insertions}) + \big(\text{Cumm. change as in (2)}\big) \\ &+ (\text{All other changes}) < 0, \end{aligned} \tag{4}$$

where the inequality comes from (2) and (3), and the fact that all other contributions to potential are strictly non-positive (intuitively this is Lemma 4.11, which is proven for

the malicious model in Lemma D.11). (4) shows that the cumulative change in potential is *negative*, which means there have been more packet transfers than is possible if all nodes behaved honestly. We find the offending node by looking at control information of type 2 to find a node that is responsible for too much potential drop (intuitively, this node is duplicating packets, causing extra packet transfers that artificially increase potential drop). This argument is formalized in Theorem D.28.

Case F3. In this case, the Sender has the complete testimony from every participating node for a transmission that failed due to Case F3. Nodes exchange signatures every time a packet is passed between them, acknowledging the packet transfer; in particular, they keep a running count of the number of packets they have sent to each neighbor, and they refuse to send/receive packets from a given neighbor until they have a current signature from this neighbor on this count (this is control information of type 1). Therefore, for any honest node N in the network, summing over these (signed) counts from all of its neighbors, the total number of packets *received* by N minus the total number of packets *sent* by N will equal the number of packets currently stored in N's buffers (up to an "off-by-one" error along each edge, due to the fact that N may not have the most recent signature from a neighbor reflecting the most recent packet sent/received along the edge). Meanwhile, summing over *all* nodes (including Sender and Receiver), the net number of packets *sent* minus the total number of packets *received* should be zero: every packet sent by some node is received by another (again the "off-by-one" error can lead to a cumulative error of up to $2n$ on this sum, since it comes from the signed counts):

$$0 \approx \sum_{N \in G^*} \big((\text{Num. packets } \textit{rec'd} \text{ by } N \text{ from all neighbors}) - (\text{Num. packets } \textit{sent} \text{ by } N \text{ to all neighbors}) \big),$$

where G^* represents the subset of nodes that participated in the transmission in question. Separating out the Sender and Receiver's contribution to this sum:

$$6n^3 \approx \sum_{N \in G^* \setminus \{S,R\}} \big((\text{Num. packets } \textit{rec'd} \text{ by } N \text{ from all neighbors}) - (\text{Num. packets } \textit{sent} \text{ by } N \text{ to all neighbors}) \big),$$

where we have used the fact that the Sender inserted all D packets and the Receiver got fewer than $D - 6n^3$ of them (otherwise R could have decoded the message, resulting in a *successful* transmission). By an averaging argument, there is some node N that satisfies

$$6n^2 \leq \big((\text{Num. packets } \textit{rec'd} \text{ by } N \text{ from all neighbors}) - (\text{Num. packets } \textit{sent} \text{ by } N \text{ to all neighbors}) \big). \tag{5}$$

As noted above, the quantity on the right-hand side of (5) represents the current number of packets N is currently storing in its buffers, and a value of $6n^2$ is more than N is

allowed to store, implying that N is corrupt. The proof of Theorem D.33 formalizes this argument.

Case F4. In this case, the Sender has the complete testimony from every participating node for a transmission that failed due to Case F4. The proof of this case is analogous to the proof of Case F3 above, except that rather than considering the total number of packets that were sent/received between each node, here the Sender looks at the number of times the packet p was exchanged between each node (available via control information of type 3). Summing over all the counts of times p was sent minus the times p was received yields zero:[33]

$$0 = \sum_{N \in G^*} \big((\text{Num. times } p \textit{ rec'd} \text{ by } N \text{ from all neighbors}) - (\text{Num. times } p \textit{ sent} \text{ by } N \text{ to all neighbors}) \big) \tag{6}$$

But the Receiver's contribution to this sum is at least *negative two* since R received p at least twice (since we are in Case F4), while the Sender's contribution to this sum is exactly one (since S inserted p exactly once):

$$-1 = \sum_{N \in G^* \setminus \{S,R\}} \big((\text{Num. times } p \textit{ rec'd} \text{ by } N \text{ from all neighbors}) - (\text{Num. times } p \textit{ sent} \text{ by } N \text{ to all neighbors}) \big)$$

Using an averaging argument, there exists some node N with

$$-1 \geq \big((\text{Num. packets } \textit{rec'd} \text{ by } N \text{ from all neighbors}) - (\text{Num. packets } \textit{sent} \text{ by } N \text{ to all neighbors}) \big),$$

which implies N *sent* p more times than N received p, something that will never be true for honest nodes. This argument is formalized in Theorem D.34. □

6. Conclusion

In this paper, we have presented 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 demonstrates that adding 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

[33] We explain in the proof of Theorem D.34 how to account for the "off-by-one" error discussed above, so that the equality in (6) is precise.

remain important open questions. The original Slide protocol[34] 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.[35]

In practice, the extra factor of n^2 in terms of memory 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 $\Theta(n^3)$ 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.

Apart from the memory costs of our protocol, the computation costs and the complexity of the protocol make it unrealistic to employ in practice. If a practical protocol is to be developed for the network model considered here, there must be work done to reduce these complexities and costs.

Another open question is how altering the assumptions made in the network model affects the optimal protocol performance that is achievable. For example, Bunn and Ostrovsky [9] explore routing in *asynchronous* networks that have no minimal *connectivity assumptions*. It would be interesting to see how performance is affected in networks that make different (combinations) of assumptions.

One final open problem is to extend our setting of end-to-end communication to the case of multiple sender/receivers. In particular, can one do better than the trivial extension of our protocol, which would add a factor of $\Theta(n^2)$ to the already burdensome memory cost of $\Theta(n^4)$ per node.

Appendix A. Pseudo-Code for the Edge-Scheduling Adversarial Protocol

In this section we present pseudo-code for the Slide protocol, which will be evaluated in the next section with respect to networks operating in the edge-scheduling adversary model of Sect. 3.1. As mentioned, the Slide protocol was developed in a series of works: [2,3,6], and [14]. The version presented here varies slightly from these other versions, but no significant changes have been made.

In what follows, let $[a..b]$ denote the integers between a and b (inclusive); i.e. $[a..b] = [a, b] \cap \mathbb{Z}$.

[34] In [14], 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.

[35] 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 $O(Ln^3 \log n)$. Similarly, if we are given a bound that the maximum degree (number of edges) for all nodes is bounded by δ, then the memory bound can be further reduced to $O(L\delta n^2 \log n)$.

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). Equivalently, P is the number of bits in each packet;
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

Fig. A.1. Pseudo-code for internal nodes' setup for the edge-scheduling adversarial model.

Sender and Receiver's Additional Setup
DEFINITION OF ADDITIONAL VARIABLES FOR SENDER:
36 $\mathcal{M} := \{m_1, m_2, \ldots\}$ = Input Stream of Messages;
37 $\kappa \in [0..D]$ = Number of packets corresponding to current codeword the sender has *knowingly* inserted;

INITIALIZATION OF SENDER'S VARIABLES:
38 ***Distribute Packets***; ## See Fig. A.4
39 $\kappa = 0$;

DEFINITION OF ADDITIONAL VARIABLES FOR RECEIVER:
40 $I_R \in [D] \times (\{0,1\}^P \cup \perp)$ = Storage Buffer to hold packets corresponding to current codeword;
41 $\kappa \in [0..D]$:= Number of packets received corresponding to current codeword;

INITIALIZATION OF RECEIVER'S VARIABLES:
42 $\kappa = 0$;
43 *Initialize* I_R; ## Sets each element of I_R to $\perp$
End Sender and Receiver's Additional Setup

Fig. A.2. Additional code for sender and receiver setup.

Transmission $\mathtt{T}$
01 **for every** $N \in G$
02 **for every** $\mathtt{t} < 2 * (3D)$ ## The factor of 2 is for the 2 stages per round
03 **if** $\mathtt{t} \pmod 2 = 0$ **then:** ## STAGE 1
04 **for every** *outgoing* **edge** $E(N, B) \in G, N \neq R, B \neq S$
05 **if** $H_{FP} = \perp$: send $(H, \perp, \perp)$; **else:** send $(H - 1, H_{FP}, FR)$;
06 receive (H_{IN}, RR); ## If nothing rec'd, set $H_{IN} = RR = \perp$
07 ***Reset Outgoing Variables***;
08 **for every** *incoming* **edge** $E(A, N) \in G, N \neq S, A \neq R$
09 send (H, RR);
10 $sb_{OUT} = 0$; $FR = \perp$;
11 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;
12 **else if** $\mathtt{t} \pmod 2 = 1$ **then:** ## STAGE 2
13 **for every** *outgoing* **edge** $E(N, B) \in G, N \neq R, B \neq S$
14 **if** $H_{IN} \neq \perp$ **then:** ## Received B's info.
15 ***Create Flagged Packet***;
16 **if** (sb=1 **or** (sb=0 **and** $H_{OUT} > H_{IN}$)) **then:**
17 ***Send Packet***;
18 **for every** *incoming* **edge** $E(A, N) \in G, N \neq S, A \neq R$
19 ***Receive Packet***;
20 **if** $N \notin \{S, R\}$ **then:** ***Re-Shuffle***;
21 **else if** $N = R$ **then:** ***Receiver Re-Shuffle***;
22 **else if** $N = S$ **then:** ***Sender Re-Shuffle***;

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

Fig. A.3. Routing rules for edge-scheduling adversarial model.

24 ***Reset Outgoing Variables***
25 **if** $d = 1$: ## N sent a packet previous round
26 $d = 0$;
27 **if** $RR = \perp$ **or** $\perp \neq FR > RR$ ## Did not receive conf. of packet receipt
28 $sb = 1$;
29 **if** $RR \neq \perp$:
30 **if** $\perp \neq FR \leq RR$: ## B rec'd most recently sent packet
31 **if** $N = S$ **then:** $\kappa = \kappa + 1$;
32 $\mathsf{OUT}[H_{FP}] = \perp$; *Fill Gap*; ## Remove $\tilde{p}$ from OUT, shifting
down packets on top of $\tilde{p}$ if necessary
33 FR, $\tilde{p}$, $H_{FP} = \perp$; $sb = 0$; $H = H - 1$;
34 **if** $\perp \neq RR < FR$ **and** $\perp \neq H_{FP} < H$: ## B did *not* receive most recently sent packet
35 *Elevate Flagged Packet*; ## Swap $\mathsf{OUT}[H]$ and $\mathsf{OUT}[H_{FP}]$; Set $H_{FP} = H$;

36 ***Create Flagged Packet***
37 **if** $sb = 0$ **and** $H > H_{IN}$: ## Normal Status, will send top packet
38 $\tilde{p} = \mathsf{OUT}[H]$; $H_{FP} = H$; $FR = \mathtt{t}$;

39 ***Send Packet***
40 $d = 1$;
41 send $(\tilde{p}, FR)$;

42 ***Receive Packet***
43 receive (p, FR);
44 **if** $H_{OUT} = \perp$: ## Did not Rec. A's height info.
45 $sb = 1$;
46 **if** $H_{GP} > H$ **or** $(H_{GP} = \perp$ **and** $H < 2n)$: $H_{GP} = H + 1$;
47 **else if** $sb_{OUT} = 1$ **or** $H_{OUT} > H$: ## A packet should have been sent
48 **if** $p = \perp$: ## Packet was not rec'd
49 $sb = 1$;
50 **if** $H_{GP} > H$ **or** $(H_{GP} = \perp$ **and** $H < 2n)$: $H_{GP} = H + 1$;
51 **else if** $RR < FR$: ## Packet was rec'd and should keep it
52 **if** $H_{GP} = \perp$: $H_{GP} = H + 1$; ## If no slot is saved for p, put it on top
53 $sb = 0$; $\mathsf{IN}[H_{GP}] = p$; $H = H + 1$; $H_{GP} = \perp$; $RR = \mathtt{t}$;
54 **else:** ## Packet was rec'd, but already had it
55 $sb = 0$; *Fill Gap*; $H_{GP} = \perp$; ## See comment about *Fill Gap* on line 57 below
56 **else:** ## A packet should NOT have been sent
57 $sb = 0$; *Fill Gap*; $H_{GP} = \perp$; ## 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 $E(N, B) \in G$, $N \neq R$, $B \neq S$:
60 **if** $H_{FP} \neq \perp$:
61 $OUT[H_{FP}] = \perp$; *Fill Gap*; ## Remove any flagged packet $\tilde{p}$ from OUT, shifting
down packets on top of $\tilde{p}$ if necessary
62 $d, sb = 0$; FR, H_{FP}, $\tilde{p} = \perp$; $H = H - 1$;
63 **for every** *incoming* edge $E(A, N) \in G$, $N \neq S$, $A \neq R$:
64 $H_{GP} = \perp$; $sb = 0$; $RR = -1$; *Fill Gap*;
65 **if** $N \neq S$, R **then:** ***Re-Shuffle***; ## Re-balance buffers at end of each transmission
66 **if** $N = S$ **then:** ***Distribute Packets***;
67 **if** $N = R$ **then:** $\kappa = 0$; *Clear* I_R; ## Set each entry of I_R to $\perp$

68 ***Distribute Packets***
69 $\kappa = 0$; For each outgoing buffer OUT: $H_{OUT} = 2n$;
70 Clear all outgoing buffers. Fill each out. buffer with new codeword packets from $\mathcal{M}$;

Fig. A.4. 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

Fig. A.5. Re-shuffle rules for both edge-scheduling *and* (node-controlling + edge-scheduling) protocols.

Appendix B. 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 the Slide protocol of Sect. 4 enjoys.

The following lemma begins to link the pseudo-code with the high-level description of what the Slide 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 B.1. *At all times* (*i.e. all lines of code in Figs.* A.3, A.4, *and* A.5) *EXCEPT when packets travel between buffers* ((A.3.32–33), (A.4.52–53), *and* (A.5.89–90)), *along any* (*directed*) *edge* $E(A, B)$ *for any pair of internal nodes* (A, B), *we have*

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* $\perp \neq H_{GP} \leq H_{IN}$, *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* $\perp \neq H_{FP} > H_{OUT}$, *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 in* IN (*i.e. non-*$\perp$ *entries of* IN), *is equal to the value of* H_{IN}.
6. *The height of* OUT, *as defined by the number of packets in* OUT (*i.e. non-null entries of* OUT), *is equal to the value of* H_{OUT}.
7. *Whenever* (A.4.53) *is reached,* $H_{GP} \in [1..2n]$ *and* $H_{IN} \in [0..2n - 1]$.
8. *Whenever* (A.3.32) *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 correct*).

Additionally, during any call to Re-Shuffle:

10. *Whenever the conditional statement on line* (A.5.74) *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 re-shuffle. 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* 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 B.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 mentioned in the hypothesis of the Lemma and lines (A.3.28), (A.3.35), (A.4.38), (A.4.46), (A.4.50), (A.4.55), (A.4.57), (A.4.61–62), (A.4.64), and (A.5.91–94). In particular, we use the induction hypothesis to argue that as long as the statement of the Lemma is true going into each of these lines, then it will remain true when the protocol leaves each of these 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 $\perp$ (A.1.29–31) and (A.1.33–35) so Statements 1–6 and 9 are true (Statements 7, 8, and 10–13 are specific to certain lines of the pseudo-code, and there is nothing to prove for these in the base case).

INDUCTION STEP We now prove that each of the above statements hold after leaving lines (A.3.28), (A.3.32–33), (A.3.35), (A.4.38), (A.4.46), (A.4.50), (A.4.52–53), (A.4.55), (A.4.57), (A.4.61–62), (A.4.64), (A.5.89–90), and (A.5.91–94), provided they held upon entering these lines.

Lines (A.3.28) Statement 3 is the only relevant statement, since only *sb* is changed on line (A.3.28). However, since *sb* is set to one on this line, there is no chance that Statement 3 becomes *false* upon leaving (A.3.28) if it was *true* upon enter this line. In other words, if Statement 3 were to be false, it would not be because of line (A.3.28).

Lines (A.3.32–33) The variables in Statements 1, 2, 5, and 7 do not change in these lines, and Statements 10–13 are not relevant here, 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 (A.3.33). Also, Statement 9 will remain valid as long as Statement 8 does, as H_{FP} is set to $\perp$ on line (A.3.33), 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 (A.3.33) 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 (A.3.32) is reached, follows immediately from Claim B.6 below together with the fact that (A.3.30) must have been satisfied to reach (A.3.32). The second part of Statement 8 is proved below.

We next prove Statement 6. Anytime lines (A.3.32–33) are reached, the decrease of one by H_{OUT} on (A.3.33) 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 (A.3.32–33), 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 OUT$[H_{FP}] \neq \perp$).

Since H_{FP} is changed (A.3.33) *after* deleting a packet (A.3.32), 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 (A.3.32) was reached. This was proven above for the first part of Statement 8.

Statement 4 follows from the argument above as follows. Upon leaving line (A.3.33), $H_{FP} = \perp$, so we must show $\mathsf{OUT}[i] \neq \perp \; \forall i \in [1..H_{OUT}]$. As was argued above, $H_{FP} \neq \perp$ when (A.3.32) is reached. If $H_{FP} > H_{OUT}$ when (A.3.32) is reached, then by the induction hypothesis applied to Statement 3, on that same line $\mathsf{OUT}[i] \neq \perp \forall i \in [1..H_{OUT} - 1]$. Then when H_{OUT} is reduced by one on (A.3.33), 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 (A.3.32) 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 (A.3.32) and the packets on top of it shifted down one if necessary, so that after (A.3.32) but before (A.3.33), we will have that $\mathsf{OUT}[i] \neq \perp \forall i \in [1..H_{OUT} - 1]$. Then when H_{OUT} is reduced by one on (A.3.33), 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 (A.3.32) is reached. In particular, the height of OUT is at least one going into (A.3.32), and then the induction hypothesis applied to Statement 6 implies that $H_{OUT} \geq 1$ when (A.3.32) is reached, and the induction hypothesis applied to Statement 9 implies that $H_{OUT} \leq 2n$ when (A.3.32) is reached.

Line (A.3.35) Since only H_{FP} and OUT are modified on (A.3.35), we need only verify Statements 3, 4, 6, and 9 remain true after leaving (A.3.35). Since H_{FP} is gets the value $max(H_{OUT}, H_{FP})$ on (A.3.35), Statement 9 will be true by the induction hypothesis (applied to Statement 9). Also, the height of OUT does not change, as (A.3.35) 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 (A.3.35), since otherwise $H_{FP} = H_{OUT}$ upon leaving (A.3.35), and Statement 3 will be vacuously true. On the other hand, if $H_{FP} > H_{OUT}$, then line (A.3.35) is not reached since (A.3.34) will be false.

In order to reach (A.3.35), $H_{FP} \neq \perp$ on (A.3.34), and so both H_{OUT} and H_{FP} are not equal to $\perp$ when (A.3.35) is entered (Claim B.6), and hence $H_{FP} \neq \perp$ upon leaving (A.3.35). Also, since (A.3.35) is only reached if $H_{FP} < H_{OUT}$ (A.3.34), we use the induction hypothesis (applied to Statement 4) to argue that before reaching (A.3.35), 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 (A.3.35), Statement 4 will remain true.

Line (A.4.38) H_{FP} is the only relevant value changed on (A.4.38), so it remains to prove the relevant parts of Statements 3, 4, and 9. We will show that whenever (A.4.38) 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 (A.4.38), 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 (A.4.38) is reached. The fact that (A.4.38) was reached means that the conditional statement on the line before (A.4.37) was satisfied, and thus OUT is in normal status ($sb = 0$) and $H_{OUT} \in [1..2n]$ (the latter is true by the induction hypothesis applied to Statement 9 with respect to H_{IN} and H_{OUT}). By the induction hypothesis (applied to Statement 3), the fact that $sb = 0$ going into (A.4.37) implies that $H_{FP} = \perp$ or $H_{FP} \leq H_{OUT}$ going into (A.4.37), and then the induction hypothesis (applied to Statement 4) says that $\mathsf{OUT}[H_{OUT}] \neq \perp$ when (A.4.38) is entered.

Lines (A.4.46) and (A.4.50) The parts of Statements 1, 2, and 9 that involve 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 (A.4.46) or (A.4.50), the conditional statement says that $H_{GP} = \perp$ or $H_{GP} > H_{IN}$ going into (A.4.46) (respectively (A.4.50)). By the induction hypothesis (applied to Statement 1), $\mathsf{IN}[i] \neq \perp$ for all $1 \leq i \leq H_{IN}$, and $\mathsf{IN}[i] = \perp$ for all $i > H_{IN}$. Therefore, since IN and H_{IN} do not change on (A.4.46) or (A.4.50), 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 (A.4.46) (respectively line (A.4.50)). If $H_{GP} > H_{IN}$ going into line (A.4.46) (respectively line (A.4.50)), 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 $H_{GP} = \perp$ 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 (A.4.52–53) Statements 1, 2, 5, 7, and 9 are the only statements that are affected by these lines. Notice that H_{GP} necessarily equals $\perp$ when leaving (A.4.53), so Statement 2 is vacuously satisfied.

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 $\perp$ when leaving line (A.4.53), we must show that $\mathsf{IN}[i] \neq \perp\ \forall i \in [1..H_{IN}]$ and $\mathsf{IN}[i] = \perp\ \forall i \in [H_{IN} + 1..2n]$ upon leaving line (A.4.53). 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 $H_{GP} = \perp$, $1 \leq H_{GP} \leq H_{IN}$, or $H_{GP} = H_{IN} + 1 \leq 2n$ when line (A.4.52) is reached. We consider each case:

- If $H_{GP} = H_{IN} + 1$ when we reach line (A.4.52), then by the induction hypothesis (applied to Statement 1) it will also be true that $\mathsf{IN}[i] \neq \perp\ \forall i \in [1..H_{IN}]$ and $\mathsf{IN}[i] = \perp\ \forall i \in [H_{IN} + 1..2n]$ when this line is reached. While on line (A.4.53), 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 (A.4.53).
- If $1 \leq H_{GP} \leq H_{IN}$ when the protocol reaches (A.4.52), then also when this line is reached we have that (by the induction hypothesis applied to Statement 2) $\mathsf{IN}[i] \neq \perp\ \forall i \in [1..H_{GP} - 1]$ and $\forall i \in [H_{GP} + 1..H_{IN} + 1]$, and $\mathsf{IN}[i] = \perp\ \forall i \in [H_{IN} + 2..2n]$ and $\mathsf{IN}[H_{GP}] = \perp$. When a packet is inserted into slot H_{GP} and H_{IN}

is increased by one on line (A.4.53), 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 $\perp$, and thus Statement 1 will hold.

- If $H_{GP} = \perp$ going into line (A.4.52), 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 B.12 states that $H_{GP} \neq \perp$ when (A.4.52) 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 (A.4.52) was reached means that (A.4.47) 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 (A.4.47), and it cannot change value between then and (A.4.52).

The first part of Statement 7 is proven in the above three bullet points. For the second part, if $sb_{OUT} = 0$ when (A.4.47) was evaluated earlier in the round, then the fact that (A.4.53) 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 (A.4.47) was evaluated, then the second part of Statement 7 follows from Statement 5 of Lemma B.12.

We now prove Statement 5. There are two relevant changes made on line (A.4.53) 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 (A.4.53) is reached, $H_{GP} \in [1..2n]$ and $\mathsf{IN}[H_{GP}] = \perp$, and therefore the net effect of (A.4.53) 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 (A.4.53) by the induction hypothesis, it will remain true upon leaving (A.4.53).

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 (A.4.52) is satisfied, then $H_{IN} < 2n$, and hence the change there does not threaten the domain of H_{GP}. Also, (A.4.53) sets H_{GP} to $\perp$, which is again in the valid domain. Meanwhile, on (A.4.53) H_{IN} is changed to $H_{IN} + 1 \leq 2n$, where the inequality follows from the induction hypothesis applied to Statement 7.

Line (A.4.55), (A.4.57), and (A.4.64) 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 (A.4.55), (A.4.57), or (A.4.64), 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 (A.4.55), (A.4.57), or (A.4.64) 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 where a "ghost packet" is being stored, and this gap will be filled when *Fill Gap* is called on (A.4.55), (A.4.57) or (A.4.64). Namely, this will shift all the packets from height $H_{GP} + 1$ through $H_{IN} + 1$ *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 (A.4.55), (A.4.57) or (A.4.64) 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 (A.4.61–62) The only relevant variables to change values on these lines are sb_{OUT}, H_{OUT}, H_{FP}, and OUT, so we need only verify that Statements 3, 4, 6, and 9 remain true after leaving (A.4.61–62). First note that $H_{FP} \neq \perp$ upon reaching (A.4.61) (since (A.4.60) must be satisfied to reach (A.4.61–62)), so the induction hypothesis (applied to Statements 3 and 4) implies that $\mathsf{OUT}[H_{FP}] \neq \perp$ when (A.4.61) is reached. Therefore, by the induction hypothesis applied to Statement 6, $H_{OUT} \geq 1$ when (A.4.61) is reached, and hence $H_{OUT} \in [1..2n]$ upon reaching (A.4.61) by the induction hypothesis (applied to Statement 9). In particular, when H_{OUT} is reduced by one on (A.4.62), we will have that $H_{OUT} \in [0..2n-1]$ upon leaving (A.4.62), as required. Also, H_{FP} will be set to $\perp$ upon leaving (A.4.62), so Statement 9 remains true.

Statement 6 also follows from the fact that $\mathsf{OUT}[H_{FP}] \neq \perp$ when (A.4.61) is reached, as follows. Since (by induction) Statement 6 was true upon reaching (A.4.61), the packet deleted from OUT on (A.4.61) is accounted for by the drop in H_{OUT} on (A.4.62).

Statement 3 is vacuously true upon leaving (A.4.62), so it remains to prove Statement 4. This argument is identical to the one used to prove Statement 4 in lines (A.3.32–33) above.

Lines (A.5.89–94) We first prove Statements 10–13, and then address Statements 1–9. We first prove that before *Shuffle Packet* is called on (A.5.77), we have that $B_F[M] \neq \perp$ and $B_T[m+1] = \perp$, from which Statement 10 follows.

- 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 lines (A.5.80) and (A.5.82) 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 (A.5.80) 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 $H_{B_F} \geq 2$ in order for (A.5.74) to be true, since $m \geq 0$ by the induction hypothesis applied to Statement 9, and then the comment on (A.5.74), together with the fact that B_F is an *outgoing* buffer, means that $H_{B_F} \geq 2$).
- If B_F is an incoming buffer and $B_F[M+1] \neq \perp$, then (A.5.82) is satisfied and M is set to $M+1$ on line (A.5.83), and thus for the new value of M, $B_F[M] \neq \perp$ after line (A.5.83).
- 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 $H_{GP} = \perp$ or $H_{GP} > H_{IN} = M$. 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 (A.5.84) will be satisfied, and hence m is set to $m-1$. Thus after line (A.5.85), $B_T[m+1] = \perp$.
- 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 (A.5.86), 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 Statement 2, and thus after m is changed to $m + 1$ on (A.5.87), 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 four 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 (A.5.90), 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 (A.5.92), 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 (A.5.94). But if H_{GP} does change value here, then the conditional statement on the previous line guarantees 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 (A.5.89), 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 (A.5.89) 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 (A.5.74) implies that the value of m (which is the height of B_T) on line (A.5.73) 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 (A.5.91), 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 (A.5.89) is not equal to H_{GP}. In the case that $H_{GP} \neq \perp$ on line (A.5.86), the value of m will change to $H_{IN} + 1$ on line (A.5.87), 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 (A.5.89). 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 (A.5.89), 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 (A.5.73) and lines (A.5.86–87) and (A.5.89). For the former part, Statements 1 and 2 will remain true provided the packet taken from B_F on line (A.5.89) is the top-most packet in B_F. Looking at

the conditional statement on line (A.5.82), 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 (A.5.89) (since in this case $M = H_{IN}$ is changed to $H_{IN}+1$ on line (A.5.83)). On the other hand, if $\mathsf{IN}[H_{IN}+1] = \perp$ on line (A.5.82), 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 (A.5.89) (since the conditional statement on line (A.5.82) will not be satisfied, and hence the value of M will not be changed on line (A.5.83)).

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 (A.5.89) 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 will be the top packet, i.e. the packet in index $B_F[H_{OUT}]$ (see lines (A.5.72), (A.5.80–81), and (A.5.89)). 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 (A.5.80) will be satisfied, shifting the value of M to $H_{OUT} - 1$ on line (A.5.81). By the induction hypothesis applied to Statement 3, this new value of M corresponds to the top-most non-flagged packet of B_F.

When B_T is an *outgoing* buffer, Statements 3 and 4 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 (A.5.84), 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 (A.5.84) fails and hence the value of m does not change on the next line) the definition of m on line (A.5.73) and line (A.5.89) show that the first free slot of B_T will be filled. On the other hand, if $B_T[H_{OUT}] = \perp$ on line (A.5.84), 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 (A.5.73), (A.5.84–85), and (A.5.89), 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 (A.5.91) and (A.5.92). To satisfy the condition on line (A.5.74), 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 (A.5.91) and (A.5.92) will guarantee the parts of Statement 9 regarding H_{OUT} and H_{IN} remain true. Also, H_{GP} remains in the appropriate domain by induction applied to Statements 9, 12, and 13. Statements 7, 8, are not relevant. □

Lemma B.2. *The domains of all of the variables in Figs.* A.1 *and* A.2 *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 (A.1.08). Each entry of OUT is initialized to $\perp$ on (A.1.33). After this point, Statement 6 of Lemma B.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 B.1) and packets have size P, the domain for OUT is as indicated.

Copy of Packet to be Sent $\tilde{p}$ (A.1.09). This is initialized to $\perp$ on (A.1.34), and is only modified afterwards on (A.4.38), (A.3.33), and (A.4.62). By Statements 3, 4, and 9 of Lemma B.1, $OUT[H] \neq \perp$ when $\tilde{p}$ is set on (A.4.38) (since $sb = 0$ must have been true on (A.4.37) in order to reach (A.4.38)), and the changes on (A.3.33) and (A.4.62) reset $\tilde{p}$ to $\perp$. Therefore, the domain of $\tilde{p}$ is as indicated.

Outgoing Status Bit sb (A.1.10). This is initialized to 0 on (A.1.35), and is only modified afterwards on lines (A.3.33), (A.3.28), and (A.4.62), all of which change sb to 0 or 1, as required.

Packet Sent Bit d (A.1.11). This is initialized to 0 on (A.1.35), and is only modified afterwards on lines (A.3.26), (A.4.40), and (A.4.62), each of which change d to 0 or 1, as required.

Flagged Round Index FR (A.1.12). This is initialized to $\perp$ on (A.1.34), and is only modified afterwards on lines (A.4.38), (A.3.33), and (A.4.62). The latter two lines reset FR to $\perp$, while (A.4.38) 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 (A.3.02), so when FR is set to $\mathtt{t}$ on (A.4.38), it will be in $[0..6D]$, as required.

Height of Outgoing Buffer H (A.1.13). This is initialized to 0 on (A.1.35). After this point, Statement 9 of Lemma B.1 above guarantees $H \in [0..2n]$, as required.

Height of Flagged Packet H_{FP} (A.1.14). Statement 9 of Lemma B.1 guarantees that H_{FP} will lie in the appropriate domain at all times.

Round Adjacent Node Last Received a Packet RR (A.1.15). This is initialized to $\perp$ on (A.1.34), and is only modified afterwards when it is received on (A.3.06), 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, and hence so will the value received on (A.3.06).

Outgoing Buffer's Value for Adjacent Node's Incoming Buffer Height H_{IN} (A.1.16). This is initialized to 0 on (A.1.35), and is only modified afterwards on line (A.3.06), where it is set to the value sent on (A.3.09) by the adjacent node, or $\perp$ in case no value was received. Since the value sent on (A.3.09) will always be between 0 and $2n$ (by Statement 9 of Lemma B.1), H_{IN} has the required domain.

Incoming Buffers IN (A.1.18). Each entry of IN is initialized to $\perp$ on (A.1.29). After this point, Statement 5 of Lemma B.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 B.1) and packets have size P, the domain for IN is as indicated.

Packet Just Received p (A.1.19). This is initialized to $\perp$ on (A.1.30), and is only modified afterwards on (A.4.43), where it either is set to the value sent on (A.4.41) or $\perp$ in the case no value was received. Since the value sent on (A.4.41) has the appropriate domain (i.e. the size of a packet, P), in either case p has the appropriate domain.

Incoming Status Bit sb (A.1.20). This is initialized to 0 on (A.1.31), and is only modified afterwards on lines (A.4.45), (A.4.49), (A.4.53), (A.4.55), (A.4.57), and (A.4.64), all of which change sb to 0 or 1 as required.

Round Received Index RR (A.1.21). This is initialized to -1 on (A.1.31), and is only modified afterwards on lines (A.4.53) and (A.4.64). 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 (A.3.02), setting $RR = \mathtt{t}$ as on (A.4.53) will put RR in $[0..6D]$ as required. Meanwhile, (A.4.64) resets RR to -1. Thus, at all times, RR is in the appropriate domain.

Height of Incoming Buffer H (A.1.22). This is initialized to 0 on (A.1.31). After this point, Statement 9 of Lemma B.1 above guarantees $H \in [0..2n]$, as required.

Height of Ghost Packet H_{GP} (A.1.23). Statement 9 of Lemma B.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} (A.1.24). This is initialized to 0 on (A.1.31), and is only modified afterwards on line (A.3.11), where it is set to be one of the values sent on (A.3.05) by the adjacent node, or $\perp$ in case no value was received. Since the value sent on (A.3.05) (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} (A.1.25). This is initialized to 0 on (A.1.31), and is only modified afterwards on lines (A.3.10) and (A.3.11). Both changes assign sb_{OUT} to '0' or '1', as required.

Incoming Buffer's Value for Adjacent Node's Flagged Round Index FR (A.1.26). This is initialized to $\perp$ on (A.1.30), and is only modified afterwards on lines (A.3.10–11) and (A.4.43). 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 (A.3.05) and (A.4.41) 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 (A.2.37). 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 (A.2.39) and (A.4.69) guarantee that $\kappa = 0$ at the outset of any transmission. The only other place κ is modified is (A.3.31) where it is incremented by one, so we must argue that (A.3.31) is reached exactly once for every packet the sender knowingly inserts. By "knowingly" inserting a packet, we mean 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 (A.4.41). By Claim B.9 below, the sender will not to try and send any other packet besides p to its neighbor until he receives confirmation of receipt for p. There are two things to show: (1) If the sender does not receive confirmation of receipt, then κ is never incremented as on (A.3.31), and (2) If the sender *does* receive confirmation of receipt, then κ is incremented *exactly once*. By "receiving confirmation of receipt," we mean that line (A.3.30) 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 B.8 below). Clearly, (1) will be true since (A.3.31) will never be reached if (A.3.30) 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 (A.3.31) is reached this round, and κ is incremented by one there. We must show κ will not be incremented due to p ever again. To see this, p will be deleted on line (A.3.32–33) of round $\mathtt{t}'$, and therefore this packet can cause the sender to reach (A.3.31) at most once.[36] Thus, at all times κ corresponds to the number of packets (corresponding to

[36] The sender's outgoing buffers are filled with (distinct) packets from $\mathcal{M}$ at the outset of each transmission (A.4.70) and during re-shuffling (A.5.96). Since S never *receives* packets (A.3.08), once a packet p has left the sender, it will never again be in any of the sender's (outgoing) buffers. Consequently, whenever $\tilde{p}$ is

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 (A.2.40). Each entry of I_R is initialized to $\perp$ on (A.2.43), after which it is only modified on lines (A.5.101) and (A.4.67). 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 (A.5.101) is reached. We use Claim B.5 below which states that for the receiver, anytime $H_{IN} > 0$, $H_{GP} = \perp$. Therefore, whenever (A.5.99) is satisfied, Statement 1 of Lemma B.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 (A.2.41). We want to show that κ always equals the number of packets corresponding to the current codeword the receiver has received so far. Lines (A.2.42) and (A.4.67) guarantee that $\kappa = 0$ at the outset of any transmission. The only other place κ is modified is (A.5.101) where it is incremented by one, so we must argue that (A.5.101) is reached exactly once for every packet (corresponding to the current codeword) that the receiver receives. By Statement 1 of Lemma B.1 and Claim B.5 below, anytime (A.5.101) is reached, IN[1] necessarily stores a packet. This packet is added to I_R on (A.5.101) and then is promptly deleted from IN on (A.5.102). By Claim B.15, the receiver will never enter (A.5.100) twice due to the same packet, and hence (A.5.101) is reached exactly once for every distinct packet corresponding to the current codeword (see comments on (A.5.100) and (A.5.104)). 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 B.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. We prove this using induction (on the round index), noting that all buffers are balanced at the outset of the protocol (lines (A.1.29) and (A.1.33)). 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 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 the Re-Shuffle Rules dictate that N should have kept re-shuffling (A.5.72–74), a contradiction.

set as on (A.4.38) *after* round $\mathtt{t}'$, $\tilde{p}$ can never be set to p, and hence on line (A.4.38), it will never be the case (after round $\mathtt{t}'$) that $\mathsf{OUT}[H_{FP}] = p$.

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 (A.5.72) guarantee that an *incoming* buffer will be selected to take a packet from. Also, if $h = h_1$, then Re-Shuffle Rules (A.5.73) guarantee that an *outgoing* buffer will be chosen to give a packet to. Therefore, in this case a packet should have been re-shuffled (A.5.74), 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 (A.5.74). □

Lemma B.4. *Every change in network potential comes from one of the following three 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 (A.4.32), (A.4.35), (A.4.53), (A.4.55), (A.4.57), (A.4.61), (A.4.64), and (A.5.89–90). 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 (A.4.53), 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 4.7). 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 B.14, there are at most $(n-2)^2$ flagged packets in the network at any given time, and each one has maximal height $2n$ (Lemma B.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 (A.4.32), (A.4.55), (A.4.57), (A.4.61), and (A.4.64) are non-positive. Also, if (A.4.35) 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 (A.4.35) 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 (A.4.35) 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 (A.5.89–90) are strictly non-positive follows from Claim B.13 below. It remains to check the cases that a packet that was transferred between two internal nodes is accepted (A.4.53). 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 (A.4.53), 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 B.13). □

Claim B.5. *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 (A.1.31). The first statement follows immediately from line (A.5.102), 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 (A.4.53). Suppose (A.4.53) 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 (A.5.102), where H is reset to zero. Therefore, H can only be non-zero between lines (A.4.53) and (A.3.21) (when *Receiver Re-Shuffle* is called) of a given round, and at these times H_{GP} is always equal to $\perp$. □

Claim B.6. *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 (A.1.10), (A.1.12), (A.1.14)). 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 (A.1.34), 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 (A.3.33), (A.4.38), and (A.4.62), 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 (A.1.34–35). 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 ((A.3.33) and (A.4.62)), while the latter is true since *sb* only changes to one on (A.3.28), which can only be reached if $FR \neq \perp$ (A.3.27), which by the first statement of this claim implies $H_{FP} \neq \perp$. □

Claim B.7. *The following two statements are true*:

1. *Anytime* sb_{OUT} *is equal to* 1 *when* Create Flagged Packet *is called on line* (A.3.15), $H_{FP} \neq \perp$.
2. *Anytime* Send Packet *is called on line* (A.3.17), *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 second 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 (A.3.16) was satisfied. Therefore, since we are in the case $sb_{OUT} = 0$ on that line (sb_{OUT} cannot change values between (A.3.12) and (A.3.16)), then $H_{OUT} > H_{IN}$. Tracing H_{IN} backwards, it was received on line (A.3.06) and represents the value of H_{IN} that was sent on line (A.3.09). Using Statement 9 of Lemma B.1, $H_{IN} \geq 0$ and hence the value of H_{OUT} on (A.3.16) must be at least one. Since H_{OUT} and H_{IN} cannot change between lines (A.3.15) and (A.3.16) 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 (A.4.37) will be satisfied and (A.4.38) 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 (A.3.33) 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 (A.3.07) 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 $\mathtt{t}_0 + 1$, it must have changed on line (A.3.28) of round $\mathtt{t}_0 + 1$ (this is the only line that sets sb_{OUT} to 1). In particular, the conditional statements on lines (A.3.25) and (A.3.27) must have been satisfied, and so d was equal to 1 on line (A.3.25) of round $\mathtt{t}_0 + 1$. Since d is reset to

zero during Stage 1 of every round (A.3.26), it must be that d was switched from 0 to 1 on line (A.4.40) 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 (A.3.17) 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 (A.4.38) of round $\mathtt{t}_0$. It remains to argue that H_{FP} does not decrease in value between round $\mathtt{t}_0$ and line (A.3.17) of round $\mathtt{t}$. But H_{FP} can only change value on lines (A.3.33), (A.3.35), and (A.4.38). 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 (A.3.33) and (A.4.38) 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 (A.3.35), which can only *increase* H_{FP}. Thus in any case, $\perp \neq H_{FP} \geq 1$ when *Send Packet* is called on (A.3.17) of round $\mathtt{t}$.

The proof of the first statement follows from the proof given in Case 2 above. □

Definition B.8. We will say that an outgoing buffer gets *confirmation of receipt* for a packet p that it sent across its adjacent edge whenever line (A.3.30) (alternatively line (C.4.46) for the Mal-Slide protocol of Sect. 5) is reached and satisfied and the packet subsequently deleted (via "$\mathsf{OUT}[H_{FP}] = \perp$ on (A.3.32)) (respectively (C.4.50)) is (a copy of) p.

Claim B.9. *Let B, p, and $\mathtt{t}$ denote either*:

- *B is the sender, p is any packet the sender is currently storing, and $\mathtt{t} = 0$ is the outset of a transmission, OR*
- *B is an internal node and p is (an instance of) a packet that is accepted by node B in round $\mathtt{t}$ (using the definition of "accepted" from Definition 4.4)*

Then:

1. *Let $\mathtt{t}'$ be the first round after*[37] *$\mathtt{t}$ 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 $\mathtt{t}'$.*
2. *If B fails to get confirmation of receipt for the packet in round $\mathtt{t} + 1$ (i.e. either RR is not received on (A.3.06) of round $\mathtt{t}' + 1$, or it is received but $RR < \mathtt{t}'$*[38]*), then* OUT *enters problem status as on (A.3.28) 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 (A.3.06) with $RR \geq \mathtt{t}'$).*
3. *From the time p is first flagged as on (A.4.38) of round $\mathtt{t}'$ through the time B does get confirmation of receipt (or through the end of the transmission, whichever comes first),* OUT *will not have any other flagged packets, i.e. $\tilde{p} = \mathsf{OUT}[H_{FP}] = p$ and $FR = \mathtt{t}'$.*

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

[38] Failing the condition on (A.3.30) technically means $RR < FR$ or $FR = \perp$; but in light of Statement 3 of this claim, $FR = \mathtt{t}'$ for all rounds between $\mathtt{t}'$ and the time B gets confirmation of receipt for p.

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 (A.4.38) 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 (A.3.15), we must have that $sb_{OUT} = 1$ on line (A.4.37) of round $\mathtt{t}'$, and hence (A.4.38) is not reached that round. In particular, when *Send Packet* is called on line (A.3.17) (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 (A.4.38) (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 (A.4.38) 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 (A.3.28), this line must have been reached in round $\widehat{\mathtt{t}}$. In particular, this implies that (A.3.25) 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 reset to zero at the end of Stage 1 of every round as on (A.3.26)). 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 (A.3.16) of round $\mathtt{t}'$ (so that *Send Packet* could be called). Since sb_{OUT}, H_{OUT}, and H_{IN} cannot change between (A.3.15) and (A.3.17) of any round, (A.4.37) will be true, and thus FR is set to $\mathtt{t}'$ on (A.4.38) of round $\mathtt{t}'$. Also, $d = 1$ after the call to *Send Packet* of round $\mathtt{t}'$ (A.4.40). 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 (A.3.25) and (A.3.27) will be satisfied, and hence OUT will enter problem status on (A.3.28) of 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 (A.3.06) 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* (A.3.06) *with* $RR \geq \mathtt{t}_0$.

Proof. OUT will certainly return to normal status by the end of the transmission (A.4.62), 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 (A.3.33) of round $\mathtt{t}'$. In particular, lines (A.3.29) and (A.3.30) were both satisfied, so OUT must have received RR on (A.3.06) earlier in round $\mathtt{t}'$, with $RR \geq FR$. If the value of FR on

line (A.3.30) equals $\mathtt{t}_0$, then the proof is complete. We show by contradiction that this must be the case.

Assume for the sake of contradiction that $FR \neq \mathtt{t}_0$ on line (A.3.30) 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 (A.3.33) and (A.4.38). 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 (A.3.33) already passed by the start of Stage 2 (which is when the subclaim asserts $FR = \mathtt{t}_0$), and (A.4.38) cannot be reached in round $\mathtt{t}$ since OUT has problem status when (A.4.37) of round $\mathtt{t}$ is reached (by hypothesis).

- Suppose FR is *first* changed from $FR = \mathtt{t}_0$ on (A.3.33) of round $\mathtt{t}''$. First note that because (A.3.33) 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 (A.3.30) earlier in round $\mathtt{t}''$. Also, since (A.3.33) 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 (A.3.30) 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 (A.3.30) of round $\mathtt{t}''$.
- Suppose FR is *first* changed from $FR = \mathtt{t}_0$ on (A.4.38) of round $\mathtt{t}''$. Then (A.4.37) 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 (A.3.33). But if (A.3.33) 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 (A.4.38) 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 (A.4.38) of round $\mathtt{t}'$. This is the case since $sb_{OUT} = 0$ on line (A.3.12) (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 (A.3.16), and therefore since none of these values change between (A.3.12) and (A.3.16), (A.4.37) 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 (A.3.32–33) of some later round,[39] 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 (A.3.33), (A.3.35), and (A.4.38); while OUT only changes values on lines (A.3.32), (A.3.35), and (A.5.89–90). Clearly the changes to each value on (A.3.35) will preserve $\mathsf{OUT}[H_{FP}] = p$, so it is enough to check the other changes. Notice that (A.3.32) is reached if and only if (A.3.33) 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 (A.4.38) can only occur if (A.4.37) is satisfied, i.e.

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

only if OUT has normal status, 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 (A.5.89–90) will preserve $\mathsf{OUT}[H_{FP}] = p$ by Statement 11 of Lemma B.1.

That $FR = \mathtt{t}'$ from (A.4.38) 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 (A.3.33) or (A.4.38), which we already proved (in the proof of the subclaim above) are not reached. □

Claim B.10. *At any time, an outgoing buffer has at most one flagged packet.*

Proof. This follows immediately from Statement 3 of Claim B.9. □

Claim B.11. *For any outgoing buffer* OUT, *if at any time its Flagged Round value FR is equal to* t, *then* OUT *necessarily called* Send Packet *on line* (A.3.17) *of round* t.

Proof. Suppose that at some point in time, *FR* is set to t. Notice that the only place *FR* assumes non-$\perp$ values is on (A.4.38), and therefore line (A.4.37) must have been satisfied in round t. Since the values for sb_{OUT}, H_{OUT}, and H_{IN} cannot change between lines (A.3.15) and (A.3.16), the statement on (A.3.16) will also be satisfied in round t, and consequently *Send Packet* will be reached in t. □

Lemma B.12. *Suppose that* $sb_{OUT} = 1$ *when line* (A.4.47) *is reached in round* t *on an edge linking buffers* OUT *and* IN. *Further suppose that* IN *does receive the communication* (p, FR) *from* OUT *on line* (A.4.43) *of* t. *Let* $\mathtt{t}_0$ *denote the round described by FR, let h denote the number of packets 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.* $\mathtt{t}_0 \neq \perp$ *and* $\mathtt{t}_0 \leq \mathtt{t}$).
2. $h > h'$.
3. OUT *sent p to* IN *on line* (A.4.41) *of round* $\mathtt{t}_0$. *Furthermore, the height of p in* OUT *when it is sent on line* (A.4.41) *of round* t *is greater than or equal to h*.
4. *If the condition statement on line* (A.4.51) *of round* 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 on* (A.4.53), *satisfies* $\perp \neq H_{GP} \leq h' + 1 \leq 2n$.
5. *If the condition statement on line* (A.4.51) *of round* t *is satisfied, then* H_{IN} *was less than* $2n$ *at the start of all rounds between* $\mathtt{t}_0$ *and* t.

Proof of Lemma B.12. We make a series of subclaims to prove the five statements of the lemma.

Subclaim 1. *The value of FR that is sent on* (A.4.41) *of round* t *is not* $\perp$.

Proof. Since (A.4.41) is reached, *Send Packet* was called on (A.3.17). By Statement 2 of Claim B.7, we have that $H_{FP} \geq 1$ when *Send Packet* is called, and in particular $H_{FP} \neq \perp$ on line (A.3.17). Since H_{FP} cannot change between (A.3.17) and (A.4.41), we have that $H_{FP} \neq \perp$ on (A.4.41), and hence $FR \neq \perp$ on this line (Claim B.6). □

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 ((A.3.33), (A.4.38), and (A.4.62)), it is clear that when $FR \neq \perp$, *FR* will always be less than or equal to the current round index. □

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

Proof. By Subclaim 2, 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 (A.4.47) of round $\mathtt{t} = \mathtt{t}_0$ is reached. Notice that sb_{OUT} had been reset to 0 on (A.3.10) of round $\mathtt{t} = \mathtt{t}_0$, so the only way it can be '1' on (A.4.47) later that round is if it is set to one on (A.3.11). This can only happen if $H_{OUT} = \perp$ or $FR > RR$. Since (A.4.47) is reached, (A.4.44) must have failed, and since H_{OUT} does not change values between the time it is received on (A.3.11) and (A.4.44), we have that $H_{OUT} \neq \perp$ on (A.3.11). Therefore, we must have that $FR > RR$ on (A.3.11) of round $\mathtt{t} = \mathtt{t}_0$.

Notice the value for *FR* here comes from the value sent by OUT on (A.3.05), and this happens *before* line (A.4.38) has been reached in round $\mathtt{t} = \mathtt{t}_0$. Therefore, the value of *FR* received on (A.3.11) 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 (A.3.30) *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 (A.4.53)). Therefore, we may apply Claim B.11 and Statement 2 of Claim B.9 to argue that *FR* will not be changed on (A.4.38) 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 (A.4.41) is reached of round $\mathtt{t}_0$. This contradicts the definition of $\mathtt{t}_0$ as the value received on line (A.4.43) 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 (A.4.43), which in turn corresponds to the value of *FR* that was sent on line (A.4.41). 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 (A.4.38), 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 (A.4.38) must have been reached when *Create Flagged Packet* was called on line (A.3.15); so in particular sb_{OUT} must have been zero on line (A.4.37) to have entered the conditional statement. Since sb_{OUT} cannot change between the start of Stage 2 and line (A.3.15) (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 (A.3.33) 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 (A.4.38), *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 $\mathtt{t}$. Finally, if FR were to change values at any point during rounds $\mathtt{t}_0 + 1$ and $\mathtt{t} - 1$, then we again would have that FR can only assume values at least $\mathtt{t}' > \mathtt{t}_0$ after this point, and 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 $\mathtt{t}$. □

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 (A.4.38) of round $\mathtt{t}_0$, this line must have been reached in $\mathtt{t}_0$. In particular, line (A.4.37) 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, (A.3.16) 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 Claim B.9. □

Subclaim 6. *The height of* p *in* OUT *when it is sent 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 (A.4.38). 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 B.9, $p = \tilde{p}$ will remain the flagged packet through the start of round $\mathtt{t}$, and $\mathsf{OUT}[H_{FP}] = p$. By Statement 11 of Lemma B.1, H_{FP} will not change during any call to re-shuffle. Indeed, since Subclaim 4 ensures that line (A.4.38) is never reached from $\mathtt{t}_0 + 1$ through the start of $\mathtt{t}$, the only place H_{FP} can change value is on (A.3.33) or (A.3.35). We know the former cannot happen between $\mathtt{t}_0 + 1$ and the start of $\mathtt{t}$, since this would imply sb_{OUT} is reset to zero on (A.3.33) 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 (A.3.35), 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 (A.4.38) (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 (A.3.16) 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 as of line (A.4.38) of round $\mathtt{t}_0$ (Statement 6 of Lemma B.1), this subclaim will follow if h' equals the value of H_{IN} as of line (A.4.38) of round $\mathtt{t}_0$. But this is true by Statement 5 of Lemma B.1, since the value of H_{IN} on line (A.3.16) comes from the value received on line (A.3.06), which in turn corresponds to the value of H_{IN} sent on line (A.3.09). □

Subclaim 8. *If the conditional statement on line* (A.4.51) *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 (A.4.51) was satisfied in round $\widetilde{\mathtt{t}}$, and in particular *RR* is set to $\widetilde{\mathtt{t}} \geq \mathtt{t}_0$ on (A.4.53). 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 $\widetilde{\mathtt{t}} \geq \mathtt{t}_0$ on (A.4.53) 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 (A.4.53)). But this contradicts the fact that $RR < FR$ on (A.4.51) of round $\mathtt{t}$, since by definition of $\mathtt{t}_0$, $FR = \mathtt{t}_0$ on (A.4.51) of round $\mathtt{t}$. □

Subclaim 9. *If the conditional statement on line* (A.4.51) *is satisfied in round* $\mathtt{t}$*, then* $RR < \mathtt{t}_0$ *between the start of* $\mathtt{t}_0$ *through line* (A.4.51) *of round* $\mathtt{t}$*. In particular, lines* (A.4.47) *and* (A.4.51) *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 ((A.1.31) and (A.4.64)). Since the only other place *RR* changes value is (A.4.53), 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 (A.4.53) was reached, which contradicts Subclaim 8. The fact that (A.4.51) will be satisfied whenever it is reached now follows immediately from Subclaim 4, since in order to reach (A.4.51), line (A.4.48) must have failed, which means the communication on line (A.4.43) was received. The fact that (A.4.47) will be satisfied whenever it is reached follows from the fact that sb_{OUT} will always be set to one on (A.3.11) 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 (A.3.11), then $FR = \mathtt{t}_0$). □

Subclaim 10. *If the conditional statement on line* (A.4.51) *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 (A.4.51) of round $\widetilde{\mathtt{t}}$ will necessarily be reached (since the conditional statement of line (A.4.44) will fail by assumption, (A.4.47) will be satisfied by Subclaim 9, and (A.4.48) will fail by assumption). However, line (A.4.53) cannot be reached in round $\widetilde{\mathtt{t}}$ (Subclaim 8 above), and therefore the conditional statement on line (A.4.51) must fail. This contradicts Subclaim 9. □

Subclaim 11. *If the conditional statement on line* (A.4.51) *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 (at least) line* (A.4.53) *of round* $\mathtt{t}$*.*

Proof. We first show that sb_{IN} will be set to one on line (A.4.45) or (A.4.49) of round $\mathtt{t}_0$. To see this, we note that if (A.4.44) fails in round $\mathtt{t}_0$, then necessarily (A.4.47) and (A.4.48) will both be satisfied. After all, (A.4.47) is satisfied (Subclaim 9), and then (A.4.48) must be true (by Subclaim 10, since we are assuming (A.4.49) failed). Thus, sb_{IN} will be set to one on line (A.4.45) or (A.4.49) of round $\mathtt{t}_0$, as claimed. Now for every round between $\mathtt{t}_0 + 1$ and $\mathtt{t}$, Subclaims 9 and 10 imply that either the conditional statement on line (A.4.44) will be satisfied, or the conditional statements on lines (A.4.47) and (A.4.48) will both be satisfied, and hence sb_{IN} can never be reset to zero since lines (A.4.53), (A.4.55), and (A.4.57) will never be reached. □

Subclaim 12. *If the conditional statement on line* (A.4.51) *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 (A.4.46) or (A.4.50) will be reached in round $\mathtt{t}_0$ (since either line (A.4.45) or (A.4.49) 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 B.1), and since H_{IN} cannot change value between the start of $\mathtt{t}_0$ and the time *Receive Packet* is called later in $\mathtt{t}_0$, we have that the value of $H_{IN} < 2n$ when either line (A.4.46) or (A.4.50) is reached. Therefore, these lines guarantee that $\perp \neq H_{GP} \leq h' + 1 \leq 2n$ after these lines (either because H_{GP} satisfied this before these lines, or it was set to $H_{IN} + 1$ on these lines). After this, there are five places H_{GP} can change its value: (A.4.46), (A.4.50), (A.4.53), (A.4.55), and (A.4.57). As in the proof of Subclaim 11, lines (A.4.53), (A.4.55), and (A.4.57) will not be reached at any point between $\mathtt{t}_0$ and $\mathtt{t}$. 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* (A.4.51) *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 on* (A.4.53), *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} (A.4.53). □

Subclaim 14. *If the condition statement on line* (A.4.51) *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 (A.4.53), and during Re-Shuffling (A.5.91–92). Between rounds $\mathtt{t}_0$ and $\mathtt{t} - 1$, line (A.4.53) 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 B.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 B.13. *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* (A.3.35).

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 are lines (A.3.32), (A.3.35), (A.4.53), (A.4.55), (A.4.57), (A.4.61), (A.4.64), (A.5.89–90), and (A.5.101–102). Of these, (A.3.35) is excluded from the claim, and the packet movement on lines (A.3.32), (A.4.55), (A.4.57), (A.4.61), (A.4.64), and (A.5.101–102) are all clearly strictly downwards. It remains to consider lines (A.4.53) and (A.5.89–90).

Case 1: The packet moved during Re-Shuffling as on (A.5.89–90). By investigating the code on these lines, we must show that $m + 1 \leq M$. This was certainly true as of line (A.5.74), but we need to make sure this did not change when *Adjust Heights* was called. The changes made to M and m on (A.5.83) and (A.5.85) will only serve to help the inequality $m + 1 \leq M$, so we need only argue the cases for when (A.5.81) and/or (A.5.87) is reached. Notice that if either line is reached, by (A.5.74) we must have (before adjusting M and m) that $M - m \geq 2$, and therefore modifying *only* $M = M - 1$ *or* $m = m + 1$ will not threaten the inequality $m + 1 \leq M$. It remains to argue that both (A.5.81) and (A.5.87) 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 (A.5.72–74) and (A.5.80) and (A.5.86)). We argue that this cannot ever happen. By Claim B.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 (it is an *incoming* buffer) and B_1 can only get smaller (it is an *outgoing* buffer) ((A.4.53) and (A.3.33) 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 (A.5.72) and (A.5.73) shows that this can never happen.

Case 2: The packet moved during Routing as on (A.4.53). In order to reach (A.4.53), the conditional statements on lines (A.4.47) and (A.4.51) must be satisfied and the statement on (A.4.48) must fail; 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 (A.4.47). Then Statements 2–4 of Lemma B.12 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 (A.4.47). For notational convenience, denote the current round (when the hypotheses of Case B hold) by `t`. First note that Statements 1 and 2 of Lemma B.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 (A.3.11) of round t), the value received for H_{OUT} on (A.3.11) is not $\perp$, and the value for FR received on (A.3.11) is either $\perp$ or satisfies $FR \leq RR$. Notice that the case $FR \leq RR$ is not possible, since then (A.4.53) would not be reached ((A.4.51) would fail). Therefore, $FR = \perp$ but $H_{OUT} \neq \perp$, and so B_2 received the communication sent by B_1 on (A.3.05) of round t, which had the first of the two possible forms. In particular, $H_{FP} = \perp$ at the outset of t, and since H_{FP} cannot change between the start of a round and line (A.4.38) of the previous round, we must have that (A.4.37) failed in round $t - 1$. By this fact and Claim B.6, B_1 had normal status when (A.3.16) was reached in round $t - 1$, and this will not be able to change in the call to *Reset Outgoing Variables* of round t because $d = 0$ (A.3.25) (since d is reset to zero every round on (A.3.26), it can only have non-zero values between line (A.4.40) of one round and line (A.3.26) 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 (A.4.37) failed in round $t - 1$ implies that (A.3.16) will also fail in round $t - 1$). Therefore, B_1 has normal status when *Create Flagged Packet* is called in round t, and in particular, H_{FP} is set to H_{OUT} on (A.4.38), i.e. the flagged packet to be transferred during t has height H_{OUT} in B_1.[40] 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 B.14. *Before* End of Transmission Adjustments *is called in any transmission* T (A.4.58–61), *any packet that was inserted into the network during transmission* 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 the details.

First notice that the statement only concerns packets corresponding to the current codeword transmission, and hence packets deleted as on (A.4.61) 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 (A.4.61) of T is reached) are: (A.3.32), (A.4.53), and (A.5.89–90).

Line (A.4.53) This line is handled by Lemma B.1, Statements 1 and 2 (see also (A.4.52)), which say that whenever a slot of an incoming buffer is filled as on line (A.4.53), it fills an empty slot, and therefore cannot correspond to removing (overwriting) p.

[40] Since a packet was necessarily transferred in t by definition of t, and since we already noted $sb_{OUT} = 0$ at the start of t, the fact that a packet was sent means that (A.3.16) was satisfied, and thus $H_{OUT} > H_{IN}$. Since the values cannot change between (A.3.15) and (A.3.16), we have that line (A.4.37) was necessarily satisfied in round t.

Lines (A.5.89–90) These lines are handled by Lemma B.1, Statement 10.

Line (A.3.32) 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 (A.3.32) is reached), and $\mathtt{t}_0$ denote the round that B first tried to send the packet to B' as on (A.4.41). By Statement 3 of Claim B.9, $\mathtt{t}_0$ is the round that $\tilde{p}$ was most recently set to p as on line (A.4.38) (note that $\mathtt{t}_0 \leq \mathtt{t}$). Since line (A.3.32) was reached in round $\mathtt{t}$, the conditional statements on lines (A.3.29) and (A.3.30) were satisfied, and so $\perp \neq RR \geq FR \neq \perp$ when those lines were reached. By Statement 3 of Claim B.9, FR will equal $\mathtt{t}_0$ when (A.3.30) is satisfied. Since in *any* round $\mathtt{t}'$, the only non-$\perp$ value that RR can ever be set to is $\mathtt{t}'$ (A.4.53), and since $RR \geq \mathtt{t}_0 = FR$ (A.3.30), it must be that (A.4.53) was reached in some round $\mathtt{t}' \in [\mathtt{t}_0, \mathtt{t}]$. In particular, B' stored a packet as on (A.4.53) of round $\mathtt{t}'$, which by Statement 3 of Claim B.9 was necessarily p. □

Claim B.15. *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 4.4) by another node.*

Proof. For any packet p, let $\#_p$ 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 $\#_p$ can ever* increase *is on line* (A.4.53).

Proof. The only way for $\#_p$ 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 (A.3.32), (A.3.35), (A.4.53), (A.4.55), (A.4.57), (A.4.61), (A.4.64), and (A.5.89). Of these, only (A.4.53) and (A.5.89) could possibly increase $\#_p$, as the others simply shift packets within a buffer and/or delete packets. In the latter case, $\#_p$ does not change by Statement 10 of Lemma B.1. □

Observation 2. *Suppose A (including $A = S$) first sends a (copy of a) packet p to B as on* (A.4.41) *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* (A.4.41) *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 B.8) in some round $\mathtt{t} \geq \mathtt{t}_0$* (A.3.32), *or at the end of the transmission as on* (A.4.61). *In the latter case, define $\mathtt{t} := 3D$ (the last round of the transmission) for Statement (c) below.*

(c) *Between* $\mathtt{t}_0$ *and line* (A.3.07) *of round* $\mathtt{t}$, *B will accept* (*a copy of*) *p from A as on* (A.4.53) *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* (A.4.53).

Proof. Statement (a) follows from Statement 3 of Claim B.9 and Statement 11 of Lemma B.1, together with the fact that lines (A.3.32) and (A.4.61) imply that the relevant copy of A will be deleted when it does get confirmation of receipt as in Definition B.8 (or the end of the transmission). By Statement 3 of Claim B.9, 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 (A.4.53) during some round $\mathtt{t}' \in [\mathtt{t}_0, \mathtt{t}]$. Then RR will be set to $\mathtt{t}'$ on (A.4.53) of round $\mathtt{t}'$, and RR cannot obtain a *smaller* index until the next transmission (A.4.53). By Statement 3 of Claim B.9, FR will remain equal to $\mathtt{t}_0$ from line (A.4.38) of round $\mathtt{t}_0$ through the time (A.3.33) of round $\mathtt{t}$ is reached. Therefore, between $\mathtt{t}' \geq \mathtt{t}_0$ and line (A.3.33) of round $\mathtt{t}$, we have that $FR = \mathtt{t}_0 \leq \mathtt{t}' \leq RR$, and hence line (A.4.51) can never be satisfied during these times, which implies (A.4.53) can never be reached again after $\mathtt{t}'$. 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: (A.3.32), (A.3.35), (A.4.53), (A.4.55), (A.4.57), (A.4.61), (A.4.64), and (A.5.89–90). Of these, only (A.4.53) and (A.5.89–90) threaten to move p or create a new copy of p. However, the first part of Observation 2(c) says that (A.4.53) can happen at most once (and is accounted for), while Statement 11 of Lemma B.1 rules out the case that the packet is re-shuffled as on (A.5.89–90). □

Observation 3. *No packet will ever be inserted* (*see Definition* 4.5) *into the network more than once. In particular, for any packet* p, $\#_p = 0$ *until the sender inserts it* (*i.e. some node accepts the packet from the sender as on* (A.4.53)), *at which point* $\#_p = 1$. *After this point, the only way* $\#_p$ *can become larger than one is if* (A.4.53) *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 (A.2.38), (A.4.66), and (A.4.68–70), and since S never receives a packet he has already inserted (S has no incoming edges (A.3.18) nor shuffles packets between buffers ((A.3.22) and (A.5.95–96)), a given packet p can only be inserted along a single 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 $\#_p$ can increase is on (A.4.53). Whenever this line is reached, the copy stored comes from the one received on (A.4.43), which in turn was sent by another node on (A.4.41). The copy sent on (A.4.41) in turn can only be set on (A.4.38) (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 $\#_p$ 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 $\#_p$ (by definition of

$\#_p$), and the fact R never sends a copy of a packet (A.3.13) and S never accepts packets (A.3.18). □

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). □

Claim B.15 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, $\#_p = 0$, and there is nothing to show. Suppose p is inserted into the network in round $\mathtt{t}_0$, so that $\#_p = 1$ by the end of $\mathtt{t}_0$ (Observation 3). Since $\#_p = 1$, 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 $\#_p$) of itself in round $\mathtt{t}$. Notice that the only time new copies of any packet can be created is on (A.4.53) (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 (A.4.53) 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 (A.4.53) of round $\mathtt{t}$ follows immediately from Observation 2. □
□

Lemma B.16. *Suppose that in round* $\mathtt{t}$, B accepts (*as in Definition 4.4*) *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* $\mathtt{t}$ (A.3.17). *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*:

$$-O + I + 1 \quad OR \quad -O \quad (\text{if } B = R) \tag{B.1}$$

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 (A.4.53) was reached by B's incoming buffer along $E(A,B)$ in round $\mathtt{t}$. Since the packet is stored at height H_{GP} (A.4.53), 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 B.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 (A.4.53) 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 (since the height of $I_{B,A}$ does not change from the start of $\mathtt{t}$ through the start of Re-Shuffling, *except* when $I_{B,A}$ receives the packet on (A.4.53)).

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 B.8), but by definition of non-duplicated potential (see Definition 4.7), this (flagged) packet will no longer count towards non-duplicated potential the instant B accepts it as on (A.4.53) of round $\mathtt{t}$. Therefore, A's non-duplicated potential will drop by the value H_{FP} has when B accepts the packet on (A.4.53) (Statement 3 of Claim B.9), which equals O since H_{FP} cannot change between the time *Send Packet* is called on (A.3.17) and the time the packet is accepted on (A.4.53). 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. □

Lemma B.17. *Let $\mathcal{C}=N_1N_2\cdots N_l$ 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*:

$$\phi \leq -O+l-1. \tag{B.2}$$

– *Otherwise*,

$$\phi \leq -O+l-2. \tag{B.3}$$

Proof. (Induction on l).

BASE CASE: $l = 2$ So $\mathcal{C} = N_1 R$.

Case 1: $O_{N_1,R}$ had a flagged packet at the start of $\mathtt{t}$ that had already been accepted by N_2. Our aim for this case is to prove (B.2) for $l = 2$. If $O < 2$, then $-O + l - 1 \geq -1 + 2 - 1 = 0$, and then (B.2) will be true by Lemma 4.11. 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 see that $RR \geq \tilde{\mathtt{t}}$ (A.4.53), and N_1 will receive this value for RR in R's stage one communication (A.3.06), (A.3.09). By Statement 3 of Claim B.9, $FR \leq \tilde{\mathtt{t}} \leq RR$, and thus lines (A.3.29–30) will be satisfied in round $\mathtt{t}$, so the flagged packet is deleted on (A.3.32–33), O gets the value $O - 1$, and sb is set to '0'. When *Create Flagged Packet* is called on (A.3.15), a new packet will be flagged since $sb = 0$ and $H_{OUT} = O - 1 \geq 1 > H_{IN} = 0$, 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 B.1). Letting I denote the height of the receiver's incoming buffer along $E(N_1, R)$, we see that $I = 0$ (Claim B.5). Therefore, $H_{OUT} > H_{IN}$, and so the flagged packet will be sent as on (A.3.17). Since R will receive and store this packet (since the edge is active and $RR < \mathtt{t} = FR$, lines (A.4.44) and (A.4.48) will fail, while lines (A.4.47) and (A.4.51) will be satisfied), we apply Lemma B.16 to argue there will be a change in non-duplication potential that is less than or equal to $-(O - 1)$, which is (B.2) (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 (B.3) for $l = 2$. If $O = 0$, then $-O + l - 2 = 0$, and (B.3) is true by Lemma 4.11. So assume $O \geq 1$. Then necessarily a packet will be sent during round $\mathtt{t}$ ((A.3.16) is necessarily satisfied since by assumption $E(N_1, R)$ is active during $\mathtt{t}$, $H_{OUT} \geq 1$ by Lemma B.1 and $H_{IN} = 0$ by Claim B.5). 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 B.6. 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 B.7 implies no packet was sent the previous round, and hence $d = 0$ at the start of $\mathtt{t}$ (d was necessarily zero as of (A.3.26) of round $\mathtt{t} - 1$, and as argued did not change to '1' on (A.4.40) 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 $H_{OUT} = O > 0 = I = H_{IN}$, (A.4.38) 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 B.8) 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 $\mathtt{t}$, $RR < \mathtt{t}_0$.[41] Meanwhile, by Statement 3 of Claim B.9, $FR = \mathtt{t}_0$ and $H_{FP} \neq \perp$ at the start of $\mathtt{t}$. Since these do not change values between the start of $\mathtt{t}$ and the time *Reset Outgoing Variables* is called in round $\mathtt{t}$, line (A.3.34) guarantees that if $H_{FP} < O$, then line (A.3.35) 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 B.16 to argue that $\phi \leq -O$, which is (B.3) (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 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* $\mathtt{t}$ (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:

$$\phi \leq -O_{N_2,N_3} + (l-1) - 1 \leq -I_{N_2,N_1} + l - 2 \leq -O_{N_1,N_2} + l - 2, \tag{B.4}$$

where the first inequality is from the induction hypothesis applied to the chain $N_2 \ldots R$, the second follows from Lemma B.3, and the third follows from the fact we are assuming $O_{N_1,N_2} \leq I_{N_2,N_1}$. Therefore, both (B.2) and (B.3) are satisfied. We may therefore assume in both cases below:

$$O_{N_1,N_2} > I_{N_2,N_1}. \tag{B.5}$$

Case 1: O_{N_1,N_2} had a flagged packet at the start of $\mathtt{t}$ that had already been 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 (B.4), we would have $\phi \leq -O_{N_1,N_2} + l - 1$, which is (B.2). Therefore, it remains to consider the case:

$$O_{N_1,N_2} \geq I_{N_2,N_1} + 2. \tag{B.6}$$

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

$$-(O_{N_1,N_2} - 1) + I_{N_2,N_1} + 1. \tag{B.7}$$

Also, when the receiving node N_2 accepts this packet as on (A.4.53), the height of the corresponding buffer increases by one on this line. We emphasize this fact for use below:

Fact 2. After the Routing Phase but before the call to *Re-Shuffle* in round $\mathtt{t}$, N_2's incoming buffer along $E(N_1, N_2)$ has height $I_{N_2,N_1} + 1$.

[41] By Statement 3 of Claim B.9, 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 (A.4.53). Since RR begins each transmission equal to -1 ((A.1.31) and (A.4.64)) and can only be changed after this on (A.4.53), necessarily $RR < \mathtt{t}_0$ through the start of $\mathtt{t}$.

Meanwhile, we may apply the induction hypothesis to the chain $\mathcal{C}' := N_2 \cdots R$, 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 $\mathtt{t}$ that had already been accepted by N_3.
(b) $-O_{N_2,N_3} + (l-1) - 2$, otherwise.

Adding these contributions to (B.7), we have

$$\begin{aligned}\phi &\le \big((-O_{N_1,N_2}+1) + I_{N_2,N_1} + 1\big) + \big(-O_{N_2,N_3} + (l-1) - x\big) \\ &= (-O_{N_1,N_2} + l - 1) + (-O_{N_2,N_3} + I_{N_2,N_1}) + (2-x), \end{aligned} \tag{B.8}$$

where $x = 1$ or 2, depending on whether we are in case (a) or (b) above. By Claim B.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 $(-O_{N_2,N_3} + I_{N_2,N_1}) + (2-x) \le 0$, regardless whether $x = 1$ or 2, and hence (B.8) implies (B.2). Also, if $x = 2$, then $(-O_{N_2,N_3} + I_{N_2,N_1}) + (2-x) \le 0$ (by Claim B.3), and hence (B.8) implies (B.2). It remains to consider the case $x = 1$ and $-O_{N_2,N_3} + I_{N_2,N_1} = 0$, in which case (B.8) becomes

$$\phi \le (-O_{N_1,N_2} + l - 1) + 1. \tag{B.9}$$

In order to obtain (B.2) from (B.9), 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 2 of this lemma) by arguing:

(a) After the Routing Phase of round $\mathtt{t}$ 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 $\mathtt{t}$, 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 $\mathtt{t}$ 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 $\mathtt{t}$, 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 we can show these two things, this case will be done, as during the first call to *Re-Shuffle* in round $\mathtt{t}$, we will have $M - m \ge (O_{N_2,N_3} + 1) - (O_{N_2,N_3} - 1) \ge 2$ (the call to *Adjust Heights* can only help this inequality since the selection process on (A.5.72–73) and the two items above guarantee (A.5.80) and (A.5.86) will both fail if reached), and consequently the re-shuffle on (A.5.89–90) will cause a drop of at least one to ϕ.[42]

[42] This drop was *not* already accounted for when we invoked the induction hypothesis, because the definition of ϕ does not count the re-shuffling in the chain's first node (however, since N_2 is actually the second node in the chain $\mathcal{C}$ being considered, it is valid to count its contributions to ϕ based on re-shuffling packets into O_{N_2,N_3}).

We first argue (a). As noted at the beginning of Case 1 of the INDUCTION STEP, Fact 2 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 $\mathtt{t}$, 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 B.3 guarantees that all of N_2's incoming buffers have height at most O_{N_2,N_3} at the start of $\mathtt{t}$; and also that all of N_2's outgoing buffers have height at most $O_{N_2,N_3} + 1$ at the start of $\mathtt{t}$. During the Routing Phase but before the Re-Shuffle Phase of $\mathtt{t}$, outgoing buffers cannot *increase* in height (A.3.33) and incoming buffers cannot increase in height by more than one (A.4.53). Therefore, after transferring packets but before Re-Shuffling in round $\mathtt{t}$, 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 (A.5.72).

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 $\mathtt{t}$ 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 (A.3.32–33) in round $\mathtt{t}$. In particular, the height of the outgoing buffer along $E(N_2, N_3)$ will drop by one on (A.3.33), 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 $\mathtt{t}$. 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 $\mathtt{t}$, argued as follows. At the start of $\mathtt{t}$, 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 B.3 guarantees that all of N_2's incoming buffers have height at least $O_{N_2,N_3} - 1$ at the start of $\mathtt{t}$; and also that all of N_2's outgoing buffers have height at least O_{N_2,N_3} at the start of $\mathtt{t}$. During the Routing Phase but before the Re-Shuffle Phase of $\mathtt{t}$, incoming buffers cannot *decrease* in height (A.4.53) and outgoing buffers can decrease in height by at most one (A.3.33). Therefore, after transferring packets but before Re-Shuffling in round $\mathtt{t}$, 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 (A.5.73).

Case 2: Either O_{N_1,N_2} has no flagged packet at the start of $\mathtt{t}$, or if so, it has not yet been accepted by N_2. By the same argument[43] 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 $\mathtt{t}$, and this packet will have height at least O_{N_1,N_2} in N_1's outgoing buffer. Therefore, by Lemma B.16, the change in non-duplicated potential due to this packet transfer is less than or equal to:

$$-O_{N_1,N_2} + I_{N_2,N_1} + 1. \tag{B.10}$$

Meanwhile, we may apply the induction hypothesis to the chain $\mathcal{C}' := N_2 \cdots R$, 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:

[43] 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 (B.5) to come to the same conclusion.

(a) $-O_{N_2,N_3} + (l-1) - 1$, if O_{N_2,N_3} had a flagged packet at the start of $\mathtt{t}$ that was already accepted by N_3.
(b) $-O_{N_2,N_3} + (l-1) - 2$, otherwise.

Adding these contributions to (B.10), we have

$$\begin{aligned}\phi &\le (-O_{N_1,N_2} + I_{N_2,N_1} + 1) + \left(-O_{N_2,N_3} + (l-1) - x\right) \\ &= (-O_{N_1,N_2} + l - 2) + (-O_{N_2,N_3} + I_{N_2,N_1}) + (2-x), \end{aligned} \tag{B.11}$$

where $x = 1$ or 2, depending on whether we are in case (a) or (b) above. Since the first term of (B.11) matches (B.3) and the latter two terms match the latter two terms of (B.8), we follow the argument of Case 1 above to conclude the proof. □

We can prove the following lemma, originally stated as Lemma 4.10 in Sect. 4.3, as a Corollary.

Lemma B.18. *If at any point in any transmission* $\mathtt{T}$*, 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* $n\beta_{\mathtt{T}}$*.*

Proof. 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 B.17. 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 B.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 B.17 says that the change in non-duplicated potential contributions of ϕ (see notation there) satisfy

$$\phi \le -O_{N_1,N_2} + l - 1 \le -2n + n - 1 < -n. \tag{B.12}$$

Since ϕ only records some of the changes to non-duplicated potential, we use Statement 3 of Lemma 4.11 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. □

Appendix C. 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 (A.5.76) (see Fig. A.5).

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

Fig. C.1. 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;
$\Omega_{\mathrm{T}} \in \{0,1\}^{O(\log n)} :=$ First parcel of Start of Transmission broadcast for transmission T;
$\beta_{\mathrm{T}} \in [0..4D] :=$ Number of rounds blocked in current transmission;
$F \in [0..n-1] :=$ Number of failed transmissions since the last corrupt node was eliminated;
$\mathcal{P}_{\mathrm{T}} \in \{0,1\}^n :=$ 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;
$\Theta_{\mathrm{T}} \in \{0,1\}^{O(k+\log n)} :=$ End of Transmission broadcast for transmission 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

Fig. C.2. 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], 6th 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:** $\beta_{\mathrm{T}} = \beta_{\mathrm{T}} + 1$;

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 $(\Omega_{\mathrm{T}}, \mathtt{T})$ in BB OR
- N has $(\Omega_{\mathrm{T}}, \mathtt{T})$ with $\Omega_{\mathrm{T}} = (|EN|, |\mathcal{B}_{\mathrm{T}}|, F, *)$, but has not yet rec'd $|EN|$ parcels as in line 200b, F parcels as in line 200c, or $|\mathcal{B}_{\mathrm{T}}|$ parcels as in line 200d OR
- N has rec'd the complete *SOT* broadcast, but every parcel has not yet passed across $E(N, B)$ OR
- N or $B \in BL$ OR
- N has $\Theta_{\mathrm{T}} \in BB$, 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 $(\Omega_{\mathrm{T}}, \mathtt{T})$ in BB OR
- N has $(\Omega_{\mathrm{T}}, \mathtt{T})$ with $\Omega_{\mathrm{T}} = (|EN|, |\mathcal{B}_{\mathrm{T}}|, F, *)$, but has not yet rec'd $|EN|$ parcels as in line 200b, F parcels as in line 200c, or $|\mathcal{B}_{\mathrm{T}}|$ parcels as in line 200d OR
- N has rec'd the complete *SOT* broadcast, but every parcel has not yet passed across $E(A, N)$ OR
- N or $A \in BL$ OR
- N has $\Theta_{\mathrm{T}} \in BB$, 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;

Fig. C.3. 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** $\perp \neq FR > RR$ ## Did not 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 $Sign(\mathtt{T}, \mathtt{t} - 2, p, FR, SIG[1], SIG[2], SIG[p])$; ## $SIG[3]$, 6th coord. sent on line 60, is kept as $SIG[2]$
63 **if** $H_{OUT} = \perp$ **or** ***Okay to Receive Packet*** is *false*: ##Did not 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 have been sent
68 ***Verify Signature One***;
69 **if** $\big($***Verify Signature One*** returns false **or** ## Signature from A was not valid, or
$p = \perp$ **or** p *not properly signed by* $S\big)$ **then:** ## Packet was not rec'd. or was not 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 T and t are correct

Fig. C.4. Routing rules for transmission T, (node-controlling + edge-scheduling) protocol (continued).

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** $\begin{cases} N \text{ has received full } SOT \text{ broadcast for } \mathtt{T} & \text{OR} \\ bp \text{ is a valid } SOT \text{ broadcast parcel rec'd in correct order (see 115 and 200)} \end{cases}$
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 testimony for $\mathtt{T}'$, see line 155
(5) A testimony parcel requested by A as indicated by α (received on line 105)
(6) An arbitrary testimony 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 testimony 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 testimony for some node $\widehat{N}$ on *BL* (see lines 150–151, 155):
121 **if** N is missing a testimony of node $\widehat{N}$:
122 Set α to the label of the node $\widehat{N}$ and the index of a testimony parcel from $\widehat{N}$ that N is missing;

Fig. C.5. Routing rules for transmission $\mathtt{T}$, (node-controlling + edge-scheduling) protocol (continued).

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 a 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 testimony parcel of $\widehat{N}$ for $\mathtt{T}''$
140 **if** $\widehat{N} = N$ has not already added its own testimony 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 *testimony* 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 Fig. C.1);
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 testimony 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 testimony to DB;
152 **else if** bp is a testimony 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 testimony 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 testimony 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 testimony 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 testimony 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 testimonies from some transmission:
168 ***Eliminate*** $\widehat{N}$; ## S can eliminate a node. See pf. of Thm. 5.2 for details

Fig. C.6. Routing rules for transmission $\mathtt{T}$, (node-controlling + edge-scheduling) protocol (continued).

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 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 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 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 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 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. 5.2 for details
(d) For each $N \in BL$, add a 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** $N \neq S$, R **then:** ***Re-Shuffle***; ## Re-balance buffers at end of each transmission
211 **if** $N = S$ **then:** ***Distribute Packets***;
212 **if** $N = R$ **then:** $\kappa = 0$; *Clear* I_R; ## Set each entry of I_R to $\perp$

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

Fig. C.7. Routing rules for transmission T, (node-controlling + edge-scheduling) protocol (continued).

Appendix D. Node-Controlling + Edge-Scheduling Protocol: Pseudo-Code Intensive Proofs

In this section, we give detailed proofs that walk through the pseudo-code of Figs. C.1–C.7 to argue very basic properties the protocol satisfies. The following lemma will relieve the need to re-prove many of the lemmas of Sect. 4.3 and Appendix B.

Lemma D.1. *The differences between the Slide protocol* (*designed for the edge-scheduling adversary network model*) *and the Mal-Slide protocol* (*for the node-controlling + edge-scheduling adversary network model*) *all fall under one of the following cases*:

1. *Extra variables in the Setup Phase and Initialization Phase*
2. *Length of transmission and codeword being transmitted in the current transmission*
3. *Need to authenticate signatures on packets*, *as on* (C.3.08) *and* (C.4.68)
4. *Need to check if it is okay to send/receive packets*, *as on* (C.4.59) *and* (C.4.63)
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 *can be perfectly simulated by the Slide protocol in the edge-scheduling adversary model by having an edge fail at the appropriate time. Also*, *differences falling under Case* 5 *affect the Mal-Slide protocol only insofar as their affect on the methods* Okay to Send/Receive Packet *and* Verify Signature One/Two. *In particular*, *the effect of these differences in Mal-Slide can again be simulated by the Slide protocol in the edge-scheduling adversary model*, *by introducing an edge failure at the appropriate time. Finally*, *between any two honest nodes*, *the authentications of Case* 3 *never fail*, *and Case* 4 *failures correspond to "wasted" rounds* (*see Definition* D.30).

Proof. Comparing the pseudo-code of Figs. A.1–A.5 to Figs. C.1–C.7, as emphasized by line numbers in *bold face*, it is clear that all differences fall under Cases 1–5 of the lemma. Also, all of the new methods in Figs. C.3–C.7 fall under Cases 3–5.

As for the differences as in Cases 3 and 4, it is clear that failing *Verify Signature One* on (C.4.86–87) is equivalent to the edge failing during Stage 2 (i.e. as if $p = \perp$ on (C.4.62) causing (C.4.69) to fail); failing *Verify Signature Two* on (C.4.89–90) is equivalent to the edge failing during Stage 1 (since this sets H_{IN} and RR to $\perp$ on (C.4.90), which is equivalent to the communication on (C.3.07) not being received); failing *Okay to Send Packet* on (C.4.59) is equivalent to the edge failing during Stage 2 (so that nothing is received on lines (C.3.22/C.4.62)); and failing *Okay to Receive Packet* on (C.4.63) is equivalent to the edge failing during Stage 1 (i.e. as if nothing is received on (C.3.13), so that $H_{OUT} = \perp$ on (C.4.63)). 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 Figs. C.3–C.7.

The second part of the last sentence is true by definition of "wasted" (see Definition D.30), and the first part follows from lines (C.3.11), (C.4.49), (C.4.60), (C.4.75), and Lemma D.19. □

Lemma D.2. *The domains of all of the variables in Figs. C.1 and C.2 are appropriate. In other words, Mal-Slide 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 Slide protocol follows from Lemmas B.1 and B.2, since all differences between the Slide protocol and the Mal-Slide protocol can be simulated by an edge failure in the edge-scheduling adversary model (Lemma D.1). So it remains to prove the lemma for the new variables appearing in Figs. C.1 and C.2 (i.e. the **bold** line numbers). The distribution of public and private keys (C.1.14) 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 (C.1.06). 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 (C.3.06), c_{bp}, α (C.3.04), and the seven signed items on (C.3.11). All of these have collective size $O(k + \log n)$ ((C.1.03–04), (C.1.21), (C.1.23), (C.1.24), (C.1.28), (C.1.29), (C.1.32), (C.1.35), and (C.1.36)). In Stage 2, this includes the transfer of the seven items on (C.4.60) and bp (C.3.15). Collectively, these have size $O(k + \log n)$ ((C.1.03–04), (C.1.17), (C.1.18), (C.1.21), and (C.1.42)).

Potential Lost Due to Re-Shuffling $SIG_{N,N}$ (C.1.12). This is initialized to zero on (C.2.46), after which it is only updated on (A.5.76), (C.4.50), (C.4.80), (C.4.82), (C.6.128), (C.6.133), (C.6.141), and (C.6.146). 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 D.18 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 (C.1.27). This is initialized to $\perp$ on (C.2.58), after which it is only updated on (C.5.93). Either no value was received on (C.5.93) (in which case $bp = \perp$), or it corresponds to the value sent on (C.5.97). As discussed below, the value of bp sent on (C.5.97) lies in the appropriate domain, and hence so does bp.

Broadcast Buffer Request α (C.1.28). This is initialized to $\perp$ on (C.2.58), after which it is only updated as in *Broadcast Parcel to Request* (C.5.117–122). On (C.5.117), α is set to $\perp$, and on (C.5.119) and (C.5.122), α includes the label of a node and a testimony parcel (see C.6.142–145), and so α is bounded by $O(k + \log n) = P$ as required.

Outgoing Verification of Broadcast Parcel Bit c_{bp} (C.1.29). This is initialized to zero on (C.2.58), after which it is only updated as on (C.4.40) and (C.5.111), where it clearly lies in the appropriate domain.

Broadcast Parcel bp to Send (C.1.42). This is initialized to $\perp$ on (C.2.52), after which it is only updated as in *Determine Broadcast Parcel to Send* (C.5.115). Looking at

the six types of broadcast parcels on line (C.5.115) and comparing the corresponding domains of these variables in Figs. C.1 and C.2, 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} (C.1.43). This is initialized to zero on (C.2.52), after which it is only updated as on (C.5.105) and (C.5.108). The value it takes on (C.5.105) will either be set to zero (if no value was received), or it will equal the value of c_{bp} sent on (C.5.101), which as shown above is either a one or zero. Meanwhile, the value it takes on (C.5.108) is zero, so at all times c_{bp} equals one or zero, as required.

First Parcel of Start of Transmission Broadcast Ω_{T} (C.2.66). This is initialized to $(0, 0, 0, 0)$ on (C.2.72) and is only changed on (C.7.174), (C.7.184), (C.7.192), (C.7.195), and (C.7.198). In all of these cases, it is clear that Ω_{T} can be expressed in $O(\log n)$ bits, as required.

Number of Rounds Blocked β_{T} (C.2.67). This is initialized to zero on (C.2.71) and is only changed on (C.3.27), (C.7.172), and (C.7.201). Notice that in the latter two cases, β_{T} is reset to zero, while β_{T} can only be incremented by one on (C.3.27) at most $4D$ times per transmission by (C.3.02). Since either line (C.7.172) or line (C.7.201) is reached at the end of every transmission (in the case a node is not eliminated as on line (C.6.163) or (C.6.168), line (C.7.201) will be reached by the call on (C.3.29)), $\beta_{\mathrm{T}} \in [0..4D]$ at all times, as required.

Number of Failed Transmissions F (C.2.68). This is initialized to zero on (C.2.71) and is only changed on (C.7.172) and (C.7.186). Notice that F is only incremented by one as on line (C.7.186) when a transmission fails. As was shown in Lemma 5.3, 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 (C.7.172).

Participating List $\mathcal{P}_{\mathrm{T}}$ (C.2.69). This is initialized to G on (C.2.73) and is only changed on (C.7.173) and (C.7.187); it is clear each time that $\mathcal{P}_{\mathrm{T}} \subseteq G$ in both places.

End of Transmission Parcel Θ_{T} (C.2.77). This is initialized to $\perp$ on (C.2.79) and is only changed on (C.7.179), where it is clear that Θ_{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 (C.1.08). 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 (C.2.46). Looking at all places information is *added* to BB (lines (C.5.106–107), (C.6.125), (C.6.127), (C.6.130), (C.6.136), (C.6.138), (C.6.142–145), (C.6.148–149), and (C.6.154–155), we see that there are 7 kinds of parcels stored in the broadcast buffer, as listed on (C.5.115) (the 7th type is to indicate which parcel to send across each edge, as on (C.5.106)). 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 (C.6.123) 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 traveled across (see e.g. (C.5.107), (C.6.125), (C.6.127), (C.6.130), (C.6.136), (C.6.138), (C.6.148), and (C.6.154)). 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 (C.1.08) is correct.

1. Receiver's End of Transmission Parcel $\Theta_{\mathtt{T}}$. This is added to a node's broadcast buffer on (C.6.125), and removed on (C.7.203). Since every internal node and the receiver will reach (C.7.203) at the end of every transmission (C.3.30), and by the inforgibility of the signature scheme, there is only one valid $\Theta_{\mathtt{T}}$ per transmission $\mathtt{T}$. 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 (C.6.127), (C.6.130), (C.6.136), and (C.6.138), and they are removed on (C.7.203). Since every internal node and the receiver will reach (C.7.203) at the end of every transmission (C.3.30), by the inforgibility of the signature scheme, for every transmission $\mathtt{T}$, *BB* stores *SOT* parcels corresponding to the current transmission only. Notice that *SOT* consists of: one parcel for 200a, $n-1$ parcels for 200b and 200d together, and up to $n-1$ parcels for 200c (see (C.7.200), and use the fact that $S \notin EN, BL$ and Lemma 5.3). Therefore, each node will have at most $2n$ *SOT* parcels 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 (C.6.148) and removed on (C.7.203). Since every internal node and the receiver will reach (C.7.203) at the end of every transmission (C.3.30), 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 (C.6.165) at most once for each node $\widehat{N} \in \mathcal{P}_{\mathtt{T}} \setminus S$ 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 (C.6.148) at most $n-1$ times. That the sender will enter line (C.6.165) at most once per node per transmission is clear since once the sender has reached (C.6.165) for some node $\widehat{N}$, the node will be removed from his blacklist on (C.6.166), and nodes are not re-added to the blacklist until the end of any transmission, as on (C.7.188). Therefore, once the sender has received some node $\widehat{N}$'s complete testimony as on (C.6.164), 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 Testimony 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 (C.5.115) in *BB* at any time.[44] This follows from the same argument as above, where it was shown that (C.6.164) 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 (C.6.155), 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

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

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 (C.6.155) 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 C.6.123).

For the sake of contradiction, suppose that BB is already storing a parcel of form $(N, \widehat{N}, \mathtt{T}'')$ when (C.6.155) 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 (C.6.155) is reached, we must have that (C.6.152) 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 (C.7.203), 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 (C.6.137–138). Therefore, all parcels in BB of form $(N, \widehat{N}, \mathtt{T}'')$ for $\mathtt{T}'' \neq \mathtt{T}'$ should have been removed from BB on line (C.6.139), yielding the desired contradiction.

5–6. TESTIMONY PARCELS. We fix an honest $N \in G$ and show that for every $\widehat{N} \in G \setminus \{S, N\}$, there are at most n testimony parcels corresponding to $\widehat{N}$ in N's broadcast buffer, and hence N's broadcast buffer will hold at most $n(n-2)$ testimony parcels at any time. Since a single node's testimony for a single transmission consists of at most n parcels (see lines (C.6.142–145)),[45] it will be enough to show that for every $\widehat{N} \in G \setminus S$, at all times N's broadcast buffer only holds testimony 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 $\widehat{N} \in G \setminus S$ and two transmissions $\mathtt{T}'$ and $\mathtt{T}''$ such that N's broadcast buffer holds at least one testimony parcel for $\widehat{N}$ from both $\mathtt{T}'$ and $\mathtt{T}''$. Notice that testimony parcels are only added to BB as on (C.6.154), and without loss of generality suppose that the testimony parcel of $\widehat{N}$ corresponding to $\mathtt{T}''$ was already in BB when one corresponding to $\mathtt{T}'$ is added to BB as on (C.6.154) of transmission $\mathtt{T}$. As was argued above, since (C.6.154) is reached in $\mathtt{T}$, (C.6.152) must have been satisfied, and since N's blacklist is cleared at the end of every transmission (C.7.203), it must be that a broadcast parcel of form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T})$, was received earlier in transmission $\mathtt{T}$. Notice that necessarily $\widehat{\mathtt{T}} = \mathtt{T}'$, since otherwise line (C.6.153) will not be satisfied. But then since $\mathtt{T}'' \neq \mathtt{T}'$, all testimony parcels of $\widehat{N}$ corresponding to transmission $\mathtt{T}''$ should have been removed from BB on (C.6.139), yielding the desired contradiction.

Now for N's *own* testimony parcels, these are added to BB on (C.6.142–145). Investigating (C.6.137), (C.6.139), and (C.6.140–145), we see that testimonies 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 (C.5.106) at any time, since the old version of α is simultaneously deleted when the new one is added on

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

(C.5.106). 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* (C.1.09–11). We treat the sender's data buffer separately below, and consider now only the data buffer of any internal node or the receiver. The data buffer is initially empty (C.2.46). 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 testimony (see item 4 on line (C.5.115)). Below, we show that these contribute at most $n-1$, $n-1$, and $(n-1)^2$ items (respectively), so that *DB* requires at most n^2 items (of size $O(\log n)$) at any time.

BLACKLIST *BL*. Each entry of *BL* is initialized to $\perp$ on (C.2.46), and *BL* is only modified on lines (C.6.134), (C.6.138), (C.6.148), and (C.7.203). *BL* is an array with $n-1$ entries, indexed by the nodes in $G \setminus S$. When a node $(\widehat{N}, \mathtt{T})$ is added to *BL* as on (C.6.138), this means that the entry of *BL* corresponding to $\widehat{N}$ is switched to be $\mathtt{T}$. When a node $(\widehat{N}, \mathtt{T})$ is removed from *BL* as on (C.6.148), this means that the entry of *BL* corresponding to $\widehat{N}$ is switched to $\perp$. Finally, when *BL* is to be cleared as on (C.6.134) and (C.7.203), this means that *BL* each entry of *BL* is set to $\perp$. Thus, in all cases, $BL \in [1..n-1] \times \{0,l\}^{O(\log n)}$ as required.

LIST OF ELIMINATED NODES *EN*. Each entry of *EN* is initialized to $\perp$ on (C.2.46), and is only modified on line (C.6.132). *EN* is an array with $n-1$ entries, indexed by the nodes in $G \setminus S$. 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 $EN \in [1..n-1] \times \{0,1\}^{O(\log n)}$ as required.

LIST OF WHICH NEIGHBOR'S KNOWS ANOTHER NODE'S TESTIMONY. Parcels of this kind are only added to or removed from *DB* on lines (C.6.134), (C.6.139), (C.6.149), and (C.6.151). We will now show that for any pair of nodes $\widehat{N}, \widehat{B} \in G \setminus S$, the data buffer of any honest node $N \in G$ will have at most one item 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 $(n-1)^2$ items. 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 (C.6.151) of $\mathtt{T}$. Since the second part of (C.6.151) is reached, the first part of (C.6.151) must have been satisfied, and since the blacklist is cleared at the end of every transmission (C.7.203), it must be that a broadcast parcel of form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T})$ was received earlier in transmission $\mathtt{T}$. Notice that necessarily $\widehat{\mathtt{T}} = \mathtt{T}'$, since otherwise line (C.6.151) 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 (C.6.139) of transmission $\mathtt{T}$, yielding the desired contradiction.

Adding these three contributions together, we see that *DB* requires at most n^2 item of size $O(\log n)$, as required.

Outgoing Signature Buffers *SIG* (C.1.17). Each outgoing signature buffer is initially empty (C.2.54), and they are only modified on (C.4.48–49), (C.6.128), (C.6.133), (C.6.141), and (C.6.146). 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 Mal-Slide is only intended to run polynomially long (in n), each entry of *SIG* is polynomial in n, as required.

Incoming Signature Buffers *SIG* (C.1.32). Each incoming signature buffer is initially empty (C.2.48), and they are only modified on (C.4.74–75), (C.6.128), (C.6.133), (C.6.141), and (C.6.146). 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 Mal-Slide is only intended to run polynomially long (in n), each entry of *SIG* is polynomial in n, as required.

Copy of Packets Buffer *COPY* (C.2.60). *COPY* is first filled on (C.2.74) and (C.7.215), with a copy of every packet corresponding to the first codeword. The only place it is modified after this is on (C.7.215), where the old copies are first deleted and then replaced with new ones.

Sender's Broadcast Buffer *BB* (C.2.61). In contrast to an internal node's broadcast buffer, the only thing the sender's broadcast buffer holds is the *Start of Transmission* broadcast (C.7.200) and the information that a node should be *removed* from the blacklist, see (C.6.165). Notice that at the outset of the protocol, *BB* only holds the Start of Transmission broadcast, which consists only of $\Omega_1 = (0, 0, 0, 0)$ (C.2.72–73). After this, the only changes made to *BB* appear on lines (C.6.165), (C.7.171), (C.7.199), and (C.7.200). Notice that for every transmission, necessarily either (C.7.171) or (C.7.199) 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 (C.7.200) and using Lemma 5.3, the former requires at most $2n$ parcels, and by the comment on (C.6.156), the latter requires at most n parcels (C.6.165). Therefore, the sender's broadcast buffer requires at most $3n$ parcels, as required.

Sender's Data Buffer *DB*, Eliminated List *EN*, and Blacklist *BL* (C.2.62–64). We will show that the sender's *DB* needs to hold at most $n^3 + n^2 + n$ items of size $O(\log n)$ at any time, and that the blacklist and list of eliminated nodes need at most n slots (of size $O(\log n)$) each. Notice that every entry of *DB* is initialized to $\perp$ on (C.2.73), after which modifications to *DB* occur only on lines (C.6.158), (C.6.160), (C.6.162), (C.6.166), (C.7.170–171), (C.7.187–188), (C.7.191), (C.7.194), (C.7.197), and (C.7.199). The sender's data buffer holds eight different kinds of information: end of transmission parcel $\Theta_{\mathtt{T}}$, testimony parcels, the participating list for up to $n - 1$ failed transmissions, the reason for failure for up to $n - 1$ failed transmissions, its own testimonies 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 B knows the complete testimony (see item 4 on line (C.5.115)).

1. End of Transmission Parcel $\Theta_{\mathtt{T}}$. Modifications to this occur only on lines (C.6.158), (C.7.171), and (C.7.199). Every transmission, the inforgibility of the signature scheme and the comment on line (C.6.156) guarantee that the sender will add $\Theta_{\mathtt{T}}$ to *DB* as on (C.6.158) at most once. Meanwhile, for every transmission, either (C.7.171) or (C.7.199) 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 nodes 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 (C.6.166), (C.7.171), and (C.7.188). "Removing" a node $\widehat{N}$ from *BL* as on (C.6.166) means changing the entry indexed by $\widehat{N}$ to $\perp$. "Clearing" the blacklist as on (C.7.171) means making every entry of the array equal to $\perp$. Finally, "adding" a node to the blacklist as on (C.7.188) means switching the entry indexed by $\widehat{N}$ to be the index of the current transmission.

3. Testimony Parcels. Modifications to this occur only on lines (C.6.162) and (C.7.171). We show in Lemma D.3 below that for any node $\widehat{N} \in \mathcal{P} \setminus S$, *DB* will hold at most $n(n-1)$ testimony parcels from $\widehat{N}$ at any time, from which we conclude that *DB* need hold at most $n(n-1)^2$ testimony parcels.
4. Participating Lists. We will view the participating list corresponding to transmission $\mathtt{T}$ as an array $[1..n] \times \{0,1\}^{O(\log n)}$, 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 (C.7.187) is reached only in the case the transmission *failed* (C.7.185), and we showed in Lemma 5.3 that there can be at most $n-1$ failed transmissions before a node is necessarily eliminated and *DB* is cleared as on (C.7.171).
5. Reason Transmissions Failed. Modifications to this occur only on lines (C.7.171), (C.7.191), (C.7.194), and (C.7.197). 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 (C.7.171), we again use Lemma 5.3 to conclude that Reason for Transmission Failures require at most $n-1$ parcels of *DB*.
6. Sender's Own Testimony. Parcels of this kind are added to *DB* on lines (C.7.191), (C.7.194), and (C.7.197), and removed on (C.7.171). 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 (C.7.171), we again use Lemma 5.3 to conclude that Reason for Transmission Failures require at most $n(n-1)$ items of size $O(\log n)$ of *DB*.
7. List of Eliminated Nodes *EN*. Modifications to this occur only on line (C.7.170). 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, and hence $EN \in [1..n] \times \{0,1\}^{O(\log n)}$.
8. The Label of a Node $\widehat{N}$ Whose Testimony is Known to B. Modifications to this occur only on lines (C.6.160), (C.6.166), and (C.7.171). We show in Claim D.5 below that for any pair of nodes $B, \widehat{N} \in G \setminus S$, *DB* will hold at most one parcel of the form $(B, \widehat{N}, \mathtt{T}')$ at any time (see e.g. (C.6.160)), 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 Figs. C.1 and C.2 have domains as indicated. □

Lemma D.3. *For any node $\widehat{N} \in G \setminus S$, the sender's data buffer will hold at most $n(n-1)$ testimony parcels from $\widehat{N}$ at any time. More specifically, for any transmission $\mathtt{T}$, let $\{\mathtt{T}_1, \ldots, \mathtt{T}_j\}$ denote the set of (earlier) transmissions such that at some point of $\mathtt{T}$, for each $1 \leq i \leq j$, the sender has at least one testimony parcel from $\widehat{N}$ corresponding to $\mathtt{T}_i$. Then $j \leq n-1$ and at this point in $\mathtt{T}$, for every $i < j$, the sender has $\widehat{N}$'s* complete *testimony for transmission $\mathtt{T}_i$.*

Proof. We first note that the first sentence follows immediately from the latter two since each testimony consists of at most n testimony parcels (C.6.142–145). Fix $\widehat{N} \in G \setminus S$ and let $\{\mathtt{T}_1, \ldots, \mathtt{T}_j\}$ be as in the lemma, ordered chronologically. We first show that $j \leq n-1$. For the sake of contradiction, suppose $j \geq n$. We first argue that for all $1 \leq i \leq j$, transmission $\mathtt{T}_i$ necessarily failed. Fix $1 \leq i \leq j$. Since *DB* contains a testimony parcel from $\widehat{N}$ for transmission $\mathtt{T}_i$, it must have been added on (C.6.162) of some transmission $\widehat{\mathtt{T}}$. Therefore, line (C.6.161) 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 (C.7.188) of transmission $\mathtt{T}_i$, which in turn implies transmission $\mathtt{T}_i$ failed (C.7.185). Therefore, transmission $\mathtt{T}_i$ failed for each $1 \leq i \leq j$.

By Lemma 5.3, there can be at most $n-1$ failed transmissions before a node is eliminated as on (C.7.169–177). Since $j \geq n$, considering failed transmissions $\{\mathtt{T}_2, \ldots, \mathtt{T}_j\}$, there must have been a transmission $\mathtt{T}_2 \leq \widetilde{\mathtt{T}} \leq \mathtt{T}_j$ such that *Eliminate N* (C.7.169) was entered in transmission $\widetilde{\mathtt{T}}$. We first argue that $\widetilde{\mathtt{T}} < \mathtt{T}_j$ as follows. If $\widetilde{\mathtt{T}} = \mathtt{T}_j$, then $(\widehat{N}, \mathtt{T}_j)$ would *not* be added to *BL* as on (C.7.188) (once the protocol enters (C.7.169), it halts until the end of the transmission (C.7.177), thus skipping (C.7.188)). But then (C.6.161) of any transmission after $\mathtt{T}_j$ cannot be satisfied for any testimony parcel corresponding to $\mathtt{T}_j$, and hence none of $\widehat{N}$'s testimony parcels corresponding to $\mathtt{T}_j$ could be added to *DB* after transmission $\mathtt{T}_j$. Similarly, none of $\widehat{N}$'s testimony parcels corresponding to $\mathtt{T}_j$ can be added to *DB* before or during transmission $\mathtt{T}_j$ by Claim D.4 below. This then contradicts the fact that at some point in time, *DB* contains one of $\widehat{N}$'s testimony parcels corresponding to $\mathtt{T}_j$.

We now see that for some transmission $\mathtt{T}_1 < \widetilde{\mathtt{T}} < \mathtt{T}_j$, *Eliminate N* is entered during $\widetilde{\mathtt{T}}$. Therefore, all of $\widehat{N}$'s testimony parcels for $\mathtt{T}_1$ are removed from *DB* on (C.7.171) and $(\widehat{N}, \mathtt{T}_1)$ is removed from *BL* on (C.7.171) of transmission $\widetilde{\mathtt{T}} < \mathtt{T}_j$. Since $\mathtt{T}_1 < \widetilde{\mathtt{T}}$, $(\widehat{N}, \mathtt{T}_1)$ will never be put on *BL* as on (C.7.188) for any transmission after $\widetilde{\mathtt{T}}$, and consequently, (C.6.161) will never be satisfied after $\widetilde{\mathtt{T}}$ for any of $\widehat{N}$'s testimony parcels from $\mathtt{T}_1$. Therefore, none of $\widehat{N}$'s testimony parcels will be put into *DB* after they are removed on (C.7.171) of $\mathtt{T}$. Meanwhile, by the end of transmission $\widetilde{\mathtt{T}} < \mathtt{T}_j$, *DB* cannot have any of $\widehat{N}$'s testimony parcels corresponding to $\mathtt{T}_j$ by Claim D.4 below. We have now contradicted the assumption that *DB* simultaneously holds some of $\widehat{N}$'s testimony 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* testimony for transmission $\mathtt{T}_i$ by the start of $\mathtt{T}$ (recall that $\mathtt{T}$ was the transmission for which there was some point of $\mathtt{T}$ where the list $\{\mathtt{T}_1, \ldots, \mathtt{T}_j\}$ existed). First note that Claim D.4 immediately implies that $\mathtt{T}_j < \mathtt{T}$. If $j = 1$, there is nothing to prove. 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 testimony parcels for $\mathtt{T}_i$, but not the entire report. Let $\widehat{\mathtt{T}}_i > \mathtt{T}_i$ denote the transmission that the testimony parcel corresponding to $\mathtt{T}_i$ was added to *DB* as on (C.6.162) ($\widehat{\mathtt{T}}_i > \mathtt{T}_i$ by Claim D.4 below). Since (C.6.162) is entered during transmission $\widehat{\mathtt{T}}_i$, it must be that (C.6.161) was satisfied, and in particular $(\widehat{N}, \mathtt{T}_i)$ was on *BL* during $\widehat{\mathtt{T}}_i$. Furthermore, since (C.6.161) cannot be reached in any transmission $\mathtt{T}$ *after* (C.7.188), we must have that $(\widehat{N}, \mathtt{T}_i)$ was on *BL* from the outset of $\widehat{\mathtt{T}}_i$. Lemma D.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 of each transmission (C.7.188), we may conclude that $(\widehat{N}, \mathtt{T}_{i+1})$ was *not* on *BL* at the start of $\widehat{\mathtt{T}}_i$. Since $(\widehat{N}, \mathtt{T}_{i+1})$ was necessarily added to *BL* as on (C.7.188) of transmission $\mathtt{T}_{i+1}$, it must be that $\widehat{N} \in \mathcal{P}_{\mathtt{T}_{i+1}}$ (C.7.187–188). In particular, $\widehat{N}$ was *not* on the blacklist by the end of $\mathtt{T}_{i+1}$ (C.7.187). Therefore, there must be some transmission $\widetilde{\mathtt{T}} \in [\widehat{\mathtt{T}}_i, \mathtt{T}_{i+1}]$ such that $(\widehat{N}, \mathtt{T}_i)$ is removed from *BL* as on (C.6.166) or (C.7.171). Both of these lead to a contradiction: the first implies the sender has $\widehat{N}$'s complete testimony for $\mathtt{T}_i$ by transmission $\mathtt{T}_{i+1} < \mathtt{T}$, contradicting the choice of $\mathtt{T}_i$. Meanwhile, (C.7.171) reached during $\widetilde{\mathtt{T}}$ implies that all testimonies corresponding to $\mathtt{T}_i$ should have been removed from *DB*, and since $\widetilde{\mathtt{T}} \geq \widehat{\mathtt{T}}_i > \mathtt{T}_i$, $(\widehat{N}, \mathtt{T}_i)$ can never be re-added to the blacklist after this point, and hence no testimony parcels corresponding to $\mathtt{T}_i$ will be added to *DB* after $\widetilde{\mathtt{T}}$ (as on (C.6.162)), contradicting the fact that *DB* contains (at least) one testimony parcel corresponding to $(\widehat{N}, \mathtt{T}_i)$ as of transmission $\mathtt{T} \geq \mathtt{T}_j \geq \widetilde{\mathtt{T}}$. □

Claim D.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 testimony 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 testimony parcels from $\mathtt{T}$ before or during $\mathtt{T}$. Since testimonies are only added to *DB* on (C.6.162), this implies that there is some transmission $\mathtt{T}' \leq \mathtt{T}$ such that (C.6.161) is satisfied at some point of $\mathtt{T}'$ *before* the *Prepare Start of Transmission Broadcast* of transmission $\mathtt{T}'$ is called ((C.6.161) cannot be reached during a transmission $\widehat{\mathtt{T}}$ in which *Prepare Start of Transmission Broadcast* has already been 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 (since (C.6.161) was satisfied in $\mathtt{T}'$). 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 $\mathtt{T}$ (C.7.188). □

Claim D.5. *For any pair of nodes $B, \widehat{N} \in G \setminus S$, the sender's data buffer will hold at most one testimony 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 C.6.156). Since parcels of this form are only added to *DB* on (C.6.160), 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 (C.6.160) of round $\mathtt{t}$. Since (C.6.160) is reached, (C.6.159) was satisfied,

so in particular $(\widehat{N}, \mathtt{T}')$ was on the sender's blacklist at the start of $\mathtt{T}$ (since between the start of $\mathtt{T}$ and the satisfaction of (C.6.159), it is not possible that a node gets added to the blacklist as on (C.7.188)). Let $\widehat{\mathtt{T}}$ denote the transmission that $(B, \widehat{N}, \mathtt{T}'')$ was (most recently) added to *DB* as on (C.6.160) (by assumption that $(B, \widehat{N}, \mathtt{T}'')$ was already in *DB* as of $\mathtt{T}$, we see that $\widehat{\mathtt{T}} \leq \mathtt{T}$), and hence $(\widehat{N}, \mathtt{T}'')$ was on the sender's blacklist at the outset of $\widehat{\mathtt{T}}$. By Lemma D.6 below, $\widehat{\mathtt{T}} < \mathtt{T}$, and also $(\widehat{N}, \mathtt{T}'')$ must have been removed from the blacklist at some point between the outset of $\widehat{\mathtt{T}}$ and the outset of $\mathtt{T}$. Notice that nodes are removed from the blacklist only on (C.6.166) and (C.7.171). However, in both of these cases, $(B, \widehat{N}, \mathtt{T}'')$ should have been removed from *DB* by the outset of $\mathtt{T}$ (see (C.6.166) and (C.7.171)), 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 D.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* (C.7.188)*, we see 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* $\mathtt{T}'$ (*up to the current time*)
3. *The sender has not received N's complete testimony corresponding to* $\mathtt{T}'$ (*as of the current time*)

Proof. The first statement of the lemma is immediate, since the only place a (node, transmission) pair is added to *BL* is on (C.7.188), and by (C.7.187), necessarily any such node is not already on the blacklist. Also, Statement (1) is immediate since (C.7.188) is only reached if the transmission fails (C.7.185). To prove Statements (2) and (3), notice that $(N, \mathtt{T}')$ is only added to the blacklist at the very end of transmission $\mathtt{T}'$ (C.7.188). In particular, if $(N, \mathtt{T}')$ is ever removed from the blacklist during some transmission[46] $\mathtt{T} > \mathtt{T}'$ as on (C.6.166) or (C.7.171), then $(N, \mathtt{T}')$ can never again appear on the blacklist (because $\mathtt{T} > \mathtt{T}'$, at any point during or after transmission $\mathtt{T}$, $(N, \mathtt{T}')$ can never again be added to *BL* as on (C.7.188) since $\mathtt{T}'$ has already passed). Therefore, if during transmission $\mathtt{T}$ a node is eliminated (as on (C.7.169–177)) or the sender receives *N*'s complete testimony of transmission $\mathtt{T}'$ (as on (C.6.164)), then *N* will be removed from the blacklist as on (C.6.166) or (C.7.171), at which point $(N, \mathtt{T}')$ can never be added to *BL* again. This proves Statements (2) and (3). □

Lemma D.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 testimony, and subsequently remove N from the blacklist as on* (C.6.164–165).

Proof. A node's testimony is formed on lines (C.6.140–145). Investigating the pseudo-code on those lines, there are at most $2(n-1)+1$ tuples (of form $(SIG[2], SIG[3], \mathtt{T}')$, $(SIG[1], \mathtt{T}')$, or $(SIG[p], \mathtt{T}')$) that comprise the testimony (one value for each of the $n-1$ incoming and outgoing buffers, and one additional value if line

[46] Since for all transmissions, lines (C.6.166) and (C.7.171) cannot be reached after line (C.7.188), we see that if $(N, \mathtt{T}')$ is removed from the blacklist as on (C.6.166) or (C.7.171) 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 transmission $\mathtt{T}'$ (C.7.188).

(C.6.143) is reached). By the assumption that the bandwidth $P = \Omega(k + \log n)$ is big enough to allow eight (value, signature) pairs (C.1.06), we can encapsulate the signatures on the incoming and outgoing buffers on any edge into one parcel, so that any testimony consists of at most n parcels. □

Lemma D.8. *If $N \in \mathcal{P}_{\mathtt{T}}$, then the sender is not missing any testimony 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 C.7.188) and the sender is still missing testimony 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 (C.7.187–88). Nodes are removed from the blacklist whenever the sender receives all of the testimony information he requested of them (C.6.164–166), or when he has just eliminated a node (C.7.171), in which case the sender no longer needs testimonies from nodes for old failed transmissions[47] (and in particular, this case falls outside the hypotheses of the theorem). Since $\mathcal{P}_{\mathtt{T}}$ is defined as non-blacklisted nodes (C.7.187), 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 (C.7.169–177), then there have not been any failed transmissions, and hence the sender is not missing any testimonies. Otherwise, let $\mathtt{T}' < \mathtt{T}$ denote the last time N was put on the blacklist, as on (C.7.188). In order for N to be put on $\mathcal{P}_{\mathtt{T}}$ on line (C.7.187) 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 testimony from N. □

Lemma D.9. *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 $1 \le i \le n - 1$ such that the sender has the complete testimony from every node on $\mathcal{P}_{\mathtt{T}_i}$.*

Proof. For the sake of contradiction, assume that transmission $\mathtt{T}_{n-1}$ marks the $(n-1)$st transmission $\{\mathtt{T}_1, \ldots, \mathtt{T}_{n-1}\}$ such that for each of these $n - 1$ failed transmissions, the sender does not have the complete testimony 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 testimony 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 testimony parcel corresponding to $\mathtt{T}_1$ by the end of transmission $\mathtt{T}_{n-1}$. Notice by Lemma D.8 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

[47] 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.

$\mathtt{T}_2$, there must be some node $N_2 \in \mathcal{P}_{\mathtt{T}_2}$ from which the sender is still missing a testimony 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 Lemma D.8 above), 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 D.23). But now we have a contradiction, since Lemma D.24 says that transmission $\mathtt{T}_{n-1}$ will not fail. □

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 D.10. *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 (C.4.59), (C.3.31–33), (C.4.63) and (C.3.35–37). Also, since A re-shuffled packets at the very end of the previous transmission (C.7.210), and as just mentioned A does not send or receive any packets until it receives the full *SOT* broadcast, no packets will be re-shuffled, as there will have been no change in the heights of the buffers since the end of the previous transmission, and thus line (A.5.74) will not be satisfied.[48] This proves the first part of the lemma. Also, since (C.4.63) will always be satisfied, (C.4.78) can never be reached, and so *RR* will remain equal to -1 ((C.2.50) and (C.7.209)). This in turn implies (C.4.46) cannot be satisfied before the full *SOT* broadcast has been received. Putting these facts together, the signature buffers cannot change as on (C.4.48–50), (C.4.74–75), (C.4.80), or (C.4.82) before A receives the complete *SOT* broadcast. Therefore, before A has received the complete *SOT* broadcast, changes to the signature buffers are confined to the ones appearing on lines (C.6.128), (C.6.133), (C.6.141), and (C.6.146), 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), see lines (C.7.174), (C.7.184), (C.7.192), (C.7.195), (C.7.198), and (C.7.200). If $|BL| = 0$, then A will clear all its entries of SIG^A

[48] If $A = R$, then re-shuffling of packets also does not occur until R receives the full *SOT* broadcast. See lines (A.5.97–102), which were necessarily reached at the end of the previous transmission.

and $SIG_{A,A}$ on (C.6.128). 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 (C.6.146). 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 T. □

In order to prove a variant of Lemma 4.10 in terms of the variables used in the Mal-Slide protocol, we will need to first restate and prove variants of Lemmas 4.11, B.16, and B.17. We begin with a variant of Lemma 4.11 (the first five statements correspond directly with Lemma 4.11, the others do not, but will be needed later):

Lemma D.11. *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*:
$$0 \leq SIG^A[3]_{S,A}. \tag{D.1}$$
2. *For* outgoing *edge $E(A, R)$, all changes to $SIG^A[3]_{A,R}$ are non-negative.*[49] *In particular, at all times*:
$$0 \leq SIG^A[3]_{A,R}. \tag{D.2}$$
3. *For* outgoing *edges $E(A, B)$, where $B \neq R$, 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*:
$$0 \leq \sum_{B \in \mathcal{P} \setminus \{A,S\}} \left(SIG^A[3]_{A,B} - SIG^A[2]_{A,B}\right). \tag{D.3}$$
4. *For* incoming *edges $E(B, A)$, where $B \neq S$, 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*:
$$0 \leq \sum_{B \in \mathcal{P} \setminus \{A,S\}} \left(SIG^A[2]_{B,A} - SIG^A[3]_{B,A}\right). \tag{D.4}$$
5. *All changes to $SIG_{A,A}$ are strictly non-negative. In particular, at all times*:
$$0 \leq SIG_{A,A}. \tag{D.5}$$
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*:
$$SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{A,B} \leq \left(4n^3 - 6n^2\right) + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{B,A}. \tag{D.6}$$

[49] $SIG^A[3]$ 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.

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*:

$$4n^2 - 8n \geq \sum_{B \in \mathcal{P} \setminus A} \left(SIG^A[1]_{B,A} - SIG^A[1]_{A,B} \right). \tag{D.7}$$

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*:[50]

$$0 \geq \sum_{B \in \mathcal{P}} \left(SIG^B[p]_{A,B} - SIG^A[p]_{B,A} \right). \tag{D.8}$$

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 (C.2.46), (C.2.48), and (C.2.54). Also, anytime the signature buffers are cleared as on (C.6.128), (C.6.133), (C.6.141), and (C.6.146), then all of the statements (except possibly 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 ((C.4.48–50), (C.4.74–75), (C.4.80), (C.4.82), and (A.5.76)), 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 (by Lemma D.10), and so we may (and do) assume this is the case in each item below. In particular, Lemma D.10 states that because we are assuming A has received the complete *SOT* broadcast for transmission $\mathtt{T}$, all of A's signature buffers will be cleared before any changes are made to them.

We now prove Statements 1–8 of the lemma.

1. Aside from being cleared, in which case (D.1) is trivially true, the only changes made to $SIG^A[3]_{S,A}$ occur on (C.4.75), where it is clear that all changes are non-negative since H_{GP} is non-negative (Statement 9 of Lemma B.1 together with Lemma D.1).
2. Aside from being cleared, in which case (D.2) is trivially true, the only changes made to $SIG^A[3]_{A,R}$ occur on (C.4.49), where it is clear that all changes are non-negative since H_{FP} is non-negative (Statement 9 of Lemma B.1 together with Lemma D.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 4.11. 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,

[50] Notice that (D.8) 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 $B \in G$, (D.8) will be satisfied if A in 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 (C.4.48–49) since here we are considering A's values along *outgoing* edge $E(A, B)$. Notice that H_{FP} cannot change between (C.3.08) of some round and (C.4.49) of the same round. Since lines (C.4.48–49) are only reached if *Verify Signature Two* accepts the signature (otherwise *RR* is set to $\perp$ on (C.4.90) and hence (C.4.45) will fail), we see that $SIG^A[2]_{A,B}$ changes by at most the value that H_{FP} had on (C.4.89) (see comments on line (C.4.88–90)), and this is the value sent/received on lines (C.3.07) and (C.3.11) and eventually stored on (C.4.48). Meanwhile, when $SIG^A[3]_{A,B}$ changes, for honest nodes it will always be an increase of H_{FP} (C.4.49), and as noted above, this value of H_{FP} is the same as it had on (C.4.89). Therefore, for honest nodes, whenever the relevant values change on (C.4.48–49), the change will respect the inequality $SIG^A[3]_{A,B} - SIG^A[2]_{A,B} \geq H_{FP} - H_{FP} = 0$.

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 4.11. 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 (C.4.74–75) since here we are considering A's values along *incoming* edge $E(B, A)$. When $SIG^A[2]_{B,A}$ changes on (C.4.74), they take on the values sent by B on (C.4.60) and received by A on (C.4.62). However, in order to reach (C.4.74), the call to Verify Signature One on (C.4.69) must have returned true. In particular, the comments on (C.4.84–86) 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 (C.4.75), for honest nodes it will always be an increase of H_{GP}. Therefore, since for honest nodes H_{GP} cannot change between (C.4.84) of some round and (C.4.75) later in the same round, whenever the relevant values change on (C.4.74–75), 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 4.11. All changes made to $SIG_{A,A}$ (aside from being cleared) occur on (A.5.76), where the change is $M + m - 1$. The fact that this quantity is strictly non-negative for honest nodes follows from Claim B.13.
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 (C.4.77) to the time it sends and gets confirmation of receipt (as in Definition B.8) for (that instance of) p.[51] We fix p and distinguish between the four possible ways p can "stay" in A, and describe the affect that each stay will have on (D.6):

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

(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 B.8. More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[3]$ as on (C.4.75), possibly some movement due to packet re-shuffling within A's buffers, and then an increase to some outgoing signature buffer $SIG^A[3]$ as on (C.4.49). 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 (C.4.75) 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 (C.4.49) is reached, so that the change to $SIG^A[3]_{A,B'}$ due to sending p is m. By Statement 3 of Claim B.9 (which remains valid since A is honest and by Lemma D.1), any packet that is eventually deleted as on (C.4.50–51) 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 D.18), we see that during this stay of p, $SIG_{A,A}$ changed by $M - m$. Therefore, considering only p's affect on the following terms, we have[52]

$$\Delta_p\left(SIG_{A,A} + SIG^A[3]_{A,B'} - SIG^A[3]_{B,A}\right) = (M - m) + m - M = 0. \quad \text{(D.9)}$$

(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 $\mathtt{T}$. More specifically, there is no incoming signature buffer $SIG^A[3]$ that changes value as on (C.4.75) due to this stay of p, but there is an increase to some outgoing signature buffer $SIG^A[3]$ as on (C.4.49). 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 $\mathtt{T}$, then considering only p's affect on the following terms, we have

$$\Delta_p\left(SIG_{A,A} + SIG^A[3]_{A,B'}\right) = (M - m) + m = M. \quad \text{(D.10)}$$

(c) 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 $SIG^A[3]$ as on (C.4.75), but there is no outgoing signature buffer $SIG^A[3]$ that changes value as on (C.4.49) 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 $\mathtt{T}$,[53] then considering only p's affect on the following terms, we have

$$\Delta_p\left(SIG_{A,A} - SIG^A[3]_{B,A}\right) = (M - m) - M = -m \leq 0. \quad \text{(D.11)}$$

[52] The notation Δ_p is meant to denote the fact that we are considering the affect of p's stay on each of the variables listed in the equation.

[53] If p was a flagged packet that was deleted as on (C.7.206), then let m denote the height p had just before it was deleted, i.e. the value of H_{FP} when (C.7.206) is reached.

(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, in this case there is no incoming signature $SIG^A[3]$ that changes value as on (C.4.75) due to this stay of p, and there is no outgoing signature buffer $SIG^A[3]$ that changes value as on (C.4.49) due to this stay of p. Letting M denote the initial height of p in one of A's buffers at the start of $\mathtt{T}$ and m the final height of p in one of A's buffers at the end of $\mathtt{T}$ (see footnote 53), then considering only p's affect on the following terms, we have

$$\Delta_p(SIG_{A,A}) = M - m \leq M. \tag{D.12}$$

We note that the above four cases cover all possibilities by Claim B.15 (which remains valid since A is honest, and Lemma D.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 $B, B' \in \mathcal{P}$.[54] 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 (D.10) and (D.12). Since these contributions to potential correspond to the initial height the packet had in one of A's buffers at the outset of $\mathtt{T}$, the sum over all such contributions cannot exceed A's potential at the outset of $\mathtt{T}$, which for an honest node A is bounded by $2(n-2)2n(2n+1)/2 < 4n^3 - 6n^2$.

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 $4n^2 - 8n$. During a transmission $\mathtt{T}$, the only places the quantities $SIG[1]$ change (aside from being cleared) are on (C.4.74) and (C.4.48). As with the proof of Statement 6 above, we consider the contribution of each packet p's stay in A:[55]

   (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 B.8. More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[1]$ as on (C.4.74) and then an increase to some outgoing signature buffer $SIG^A[1]$ as on (C.4.48). 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 (C.4.84–86) and (C.4.88–90)), we see that $SIG^A[1]_{B,A}$ will increase by 1 on (C.4.74) 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 (C.4.48). Therefore, considering only p's affect on the following terms, we have

$$\Delta_p\left(SIG^A[1]_{B,A} - SIG^A[1]_{A,B'}\right) = 1 - 1 = 0. \tag{D.13}$$

[54] Since A is honest, it will never send/receive packets from any node not in $\mathcal{P}$ by Lemma D.20.

[55] 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 (C.4.59–60) and (C.3.11). Therefore, there are only two cases to consider.

(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 $SIG^A[1]$ as on (C.4.74), but there is no outgoing signature buffer $SIG^A[1]$ that changes value as on (C.4.48) due to this stay of p. Using the notation from (a) above, then considering only p's affect on the following terms, we have

$$\Delta_p\big(SIG^A[1]_{B,A}\big) = 1. \tag{D.14}$$

We note that the above two cases cover all possibilities by Claim B.15 (which remains valid since A is honest, see Lemma D.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 in $\mathcal{P}$. Notice that the only non-zero contributions come from packets stays as in (D.14), 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 B.15). Note that this is the only place in the lemma that depends on testimonies *not* originating from A ($SIG^B[p]$ is a testimony parcel from B). A priori, there is the danger that a corrupt B can return a faulty testimony, thereby falsely implicating A. However, because $SIG^B[p]_{A,B}$ includes a valid signature from A (C.4.74), the inforgibility of the signature scheme guarantees that the only way a corrupt node B can falsely implicate 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 (C.4.74), which comes from the value received on (C.4.62), corresponding to the value sent on (C.4.60)), 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 lemma. Also notice that (other than out-dated signatures) the only place B gets valid signatures from A is on (C.4.62), and this value is one higher than the value that A itself is recording (C.4.60) until A updates $SIG^A[p]_{A,B}$ on (C.4.48). We argue in case (b) below, that whenever B has received an updated $SIG^B[p]_{A,B}$ as on (C.4.74) but A has not yet updated $SIG^A[p]_{A,B}$ as on (C.4.48) (and so these two values differ by one), then Case (b) will contribute -1 to the sum in (D.8), 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 (D.8) by using $SIG^A[p]_{A,B}$ instead of $SIG^B[p]_{A,B}$.

During a transmission $\mathtt{T}$, the only places the quantities $SIG[p]$ change (aside from being cleared) are on (C.4.74) and (C.4.48). As with the proof of Statements 6 and 7 above, we consider the contribution of each packet p's stay in A:[56]

(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 B.8.

[56] 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 (C.4.59–60) and (C.3.11). Therefore, there are only two cases to consider.

More specifically, the stay includes an increase to some incoming signature buffer $SIG^A[p]$ as on (C.4.74) and then an increase to some outgoing signature buffer $SIG^A[p]$ as on (C.4.48). 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 (C.4.84–86) and (C.4.88–90)), we see that $SIG^A[p]_{B,A}$ will increase by 1 on (C.4.74) 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 (C.4.48). Therefore, considering only p's affect on the following terms, we have

$$\Delta_p\left(SIG^A[p]_{A,B'} - SIG^A[p]_{B,A}\right) = 1 - 1 = 0. \tag{D.15}$$

(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 (C.4.74), but there is no outgoing signature buffer $SIG^A[p]$ that changes value as on (C.4.48) due to this stay of p. Using the notation from (a) above, then considering only p's affect on the following terms, we have

$$\Delta_p\left(-SIG^A[1]_{B,A}\right) = -1. \tag{D.16}$$

We note that the above two cases cover all possibilities by Claim B.15 (which remains valid since A is honest, see Lemma D.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 in $\mathcal{P}$ (note that it is enough to consider only neighbors in $\mathcal{P}$ by Lemma D.20). Notice that (D.15) does not contribute anything, so we have

$$\sum_{B\in\mathcal{P}}\left(SIG^A[p]_{A,B} - SIG^A[p]_{B,A}\right) = -x, \tag{D.17}$$

where x is the number of times Case (b) occurs. Notice that (D.8) concerns $SIG^B[p]_{A,B}$ (as opposed to $SIG^A[p]_{A,B}$). However, since B cannot report values of $SIG^B[p]_{A,B}$ from previous transmissions,[57] the only inaccurate value that B can report in its testimony parcel concerning $SIG^B[p]_{A,B}$ is by using an older value from $\mathtt{T}$. As discussed above, cheating in this manner only serves to help (D.8). On the other hand, if B does report the valid value for $SIG^B[p]_{A,B}$ (i.e. not outdated), then Lemma D.19 guarantees that $SIG^B[p]_{A,B} - SIG^A[p]_{A,B} \leq 1$, with equality if $SIG^B[p]_{A,B}$ has been updated as on (C.4.74) and $SIG^A[p]_{A,B}$ has not yet been updated after this point as on (C.4.48). Notice that every time this happens, we fall under Case (b) above, and in particular it can happen at most x

[57] 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 all of B's signatures from A from old transmissions will not be valid.

times (see definition of x above). Therefore:

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

which is (D.8).

All statements of the theorem have now been proven. □

We now prove a variant of Lemma B.16.

Lemma D.12. *Suppose that $A, B \in G$ are both* honest *nodes, and that in round* t, B accepts (*as in Definition 4.4*) *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 (C.3.20). *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*

$$\varphi_B := SIG^B[2]_{A,B} - SIG^B[3]_{A,B} \tag{D.18}$$

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*:

$$\Delta\varphi_B \ge H - I - 1 \quad OR \quad \Delta\varphi_B \ge H \quad (if\ B = R). \tag{D.19}$$

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 (C.4.77) was reached in round t, and hence so was (C.4.74–75). In particular, $SIG^B[3]_{A,B}$ will increase by H_{GP} on (C.4.75) (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 B.1 (which remain valid since B is honest by Lemma D.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 (C.4.77) 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 (C.4.74) to whatever value B received on (C.4.62) (as sent by A on (C.4.60) earlier in the round). Since A is honest, this value is H_{FP} larger than A's current value in $SIG^A[3]_{A,B}$ (C.4.60). By Lemma D.19, 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* of t (*before* B has accepted the packet). 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 (C.4.60) in round t (by definition of H and Statement 3 of Claim B.9). Since these are the only places $SIG^B[3]_{A,B}$ and $SIG^B[2]_{A,B}$ change, we see that $\Delta\varphi_B = H - H_{GP} \ge H - I - 1$, as desired (if $B = R$, then $\Delta\varphi_B = H$). □

The following is a variant of Lemma B.17.

Lemma D.13. *Let $\mathcal{C} = N_1 N_2 \ldots N_l$ 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 $E(N_i, N_{i+1})$, $1 \leq i < l$ are* active *for the entire round. Let $\Delta\phi$ denote the following changes during round* $\mathtt{t}$:

1. *For $1 \leq i < l$, changes to φ_{N_i} (see notation of Lemma D.12),*
2. *For $1 < i < l$, changes to SIG_{N_i,N_i} as on (A.5.76), when B_T is an outgoing buffer*

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 $\mathtt{t}$*, 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*:

$$\Delta\phi \geq O - l + 1. \tag{D.20}$$

– *Otherwise*,

$$\Delta\phi \geq O - l + 2. \tag{D.21}$$

Proof. Since A and B are honest, we use Lemma D.1 (since the present lemma excludes *wasted* rounds) and then follow exactly the proof of the analogous claim for the edge-scheduling model (Lemma B.17). 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 D.18), and using Lemma D.11 in place of Lemma 4.11, and Lemma D.12 in place of Lemma B.16. □

Lemma D.14. *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*:

$$n\left(\beta_{\mathtt{T}} - 4n^3\right) < \sum_{A \in \mathcal{H} \setminus S} SIG_{A,A} + \sum_{A \in \mathcal{H} \setminus S} \sum_{B \in \mathcal{P} \setminus \{A,S\}} \left(SIG^A[2]_{B,A} - SIG^A[3]_{B,A}\right). \tag{D.22}$$

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 D.13. 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 D.1), and then by Lemma B.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 D.13 says that the change of $\Delta\phi$ (see notation there) in round $\mathtt{t}$ satisfies

$$\Delta\phi \geq O_{N_1,N_2} - l + 1 \geq 2n - n + 1 > n. \tag{D.23}$$

Since $\Delta\phi$ only records some of the changes to the signature buffers, we use Lemma D.11 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 D.31). □

Lemma D.15. *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*[58] *which is corrupt:*[59]

$$\begin{aligned} &1. \quad SIG^B[2]_{A,B} \leq SIG^A[3]_{A,B} + 2n, \\ &2. \quad SIG^A[3]_{S,A} - SIG^S[2]_{S,A} \leq 2n, \\ &3. \quad \left|SIG^A[1]_{B,A} - SIG^B[1]_{B,A}\right| \leq 1 \quad \textit{and} \quad \left|SIG^A[1]_{A,B} - SIG^B[1]_{A,B}\right| \leq 1. \end{aligned} \tag{D.24}$$

Proof. As in the first paragraph of the proof of Lemma D.11, we may assume that both A and B have received the full *Start of Transmission* broadcast for $\mathtt{T}$, 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 $\mathtt{T}$. We prove each statement separately:

1. That either A or B is necessarily corrupt follows from Lemma D.19. 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 $\mathtt{T}$ (otherwise, they either would not have been accepted as a valid testimony parcel on (C.6.161), or a node will be eliminated as on C.6.163). 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 C.4.49). Similarly, other than being cleared upon receipt of the *SOT* broadcast, $SIG^B[2]_{A,B}$ is only updated on (C.4.74), and tracing this backwards, this comes from the value received on (C.4.62) which in turn was sent on (C.4.60). 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 (C.4.60). 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 (C.4.60) of some round that is *smaller* than a value it sent for $SIG^A[3]_{A,B}$ on (C.4.60) of some earlier round. Therefore, since the value A is supposed to send B on (C.4.60) is $SIG^A[3]_{A,B} + H_{FP} \leq SIG^A[3]_{A,B} + 2n$ (the inequality follows from Statement 9 of Lemma B.1 and Lemma D.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 D.19 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-labeling. We therefore only consider

[58] 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.

[59] The values of the quantities SIG^B and SIG^A all correspond to a common transmission $\mathtt{T}$ and refer to values the sender has received in the form of testimonies for $\mathtt{T}$ as on (C.6.161).

the second inequality of Statement 3. That either A or B is necessarily corrupt follows from Lemma D.19. It remains to show that the sender can identify a node that is necessarily corrupt. As 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 testimony parcel on (C.6.161), or a node will be eliminated as on C.6.163). 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 ((C.3.11) and (C.4.60)). 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 testimony 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 testimony. 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 (C.3.11) 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 did 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 to the sender in its testimony should be the same as the value B sent to A on (C.3.11) of round $\mathtt{t}_A$, which as noted above equals the value of $SIG^A[1]_{A,B}$ that A returned in its testimony. 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 $\mathtt{T}$ (since we are in Case 2, we may assume 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 (C.4.60). 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 (C.4.60) should be exactly one 1 more than the value stored in $SIG^A[1]_{A,B}$ as of line (C.3.07) of round $\mathtt{t}_A$, the latter of which was returned by A in its testimony (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 D.16. *If there exists a node $A \in G$ such that*

$$4n^3 - 4n^2 < SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B} - SIG^A[3]_{B,A}, \tag{D.25}$$

then either a node can be eliminated as in Statement 1 *of Lemma* D.15 *or as in Statement* 6 *of Lemma* D.11.

Proof. Suppose no node can be eliminated because of Statement 1 of Lemma D.15, so that for all $B \in G$:

$$SIG^B[2]_{A,B} \leq SIG^A[3]_{A,B} + 2n. \tag{D.26}$$

Then if (D.25) is true, we have

$$\begin{aligned} 4n^3 - 4n^2 &< SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B} - SIG^A[3]_{B,A} \\ &\leq SIG_{A,A} + 2n^2 + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{A,B} - SIG^A[3]_{B,A}, \end{aligned} \tag{D.27}$$

where the second inequality follows from applying (D.26) to each term of the sum. Therefore, A can be eliminated by Statement 6 of Lemma D.11. □

Corollary D.17. *In the case a transmission fails as in F*2*, 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*:[60]

$$\sum_{A \in \mathcal{P} \setminus S} SIG^A[3]_{S,A} < 2nD + 2n^2. \tag{D.28}$$

Proof. If the inequality in Statement 2 of Lemma D.15 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 D.15 holds for every $A \in \mathcal{P} \setminus S$. The corollary will be a consequence of the following observation:

Observation. *If a transmission* $\mathtt{T}$ *fails as in F*2*, then*:

$$\sum_{A \in \mathcal{P} \setminus S} SIG^S[2]_{S,A} < 2nD. \tag{D.29}$$

Proof. Let $\kappa_{\mathtt{T}}$ denote the value that κ had at the end of $\mathtt{T}$. Then a transmission falling under F2 means that $\kappa_{\mathtt{T}}$ is less than D (C.7.193–194). The structure of this proof will be prove the following facts:

[60] The values of the quantities SIG^A correspond to some transmission $\mathtt{T}$ and refer to values the sender has received in the form of testimonies for $\mathtt{T}$ as on (C.6.161).

Fact 3. For any $A \in \mathcal{P} \setminus S$, anytime $SIG^S[2]_{S,A}$ is updated as on (C.4.48), it will always be the case that $2n * SIG^S[1]_{S,A} \geq SIG^S[2]_{S,A}$. In particular, the final value for $SIG^S[2]_{S,A}$ at the end of $\mathtt{T}$ is less than or equal to $2n$ times the final value for $SIG^S[1]_{S,A}$.

Fact 4. At the end of $\mathtt{T}$: $\sum_{A \in \mathcal{P} \setminus S} SIG^S[1]_{S,A} = \kappa_{\mathtt{T}}$.

Before proving these facts, notice that they imply:

$$\sum_{A \in \mathcal{P} \setminus S} SIG^S[2]_{S,A} \leq \sum_{A \in \mathcal{P} \setminus S} 2n * SIG^S[1]_{S,A} = 2n\kappa_{\mathtt{T}} < 2nD \tag{D.30}$$

as required.

Fact 3 is immediate, since for any $A \in \mathcal{P} \setminus S$, whenever $SIG^S[2]_{S,A}$ is updated as on (C.4.48), the statement on (C.4.45) must have been satisfied, and so the statement on (C.4.89) 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 B.1 and Lemma D.1 (see comments on lines (C.4.88–90)). Fact 4 is also immediate, as κ and $SIG^S[1]_{S,A}$ all start the transmission with value zero (or $\perp$) by lines (C.2.54), (C.2.70), (C.7.199), and (C.7.214), and then κ is incremented by one on line (C.4.47) 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 (C.4.48) (as already argued, changes to $SIG^S[1]_{S,A}$ as on (C.4.48) are always increments of one, see e.g. the comments on lines (C.4.88–90)). □

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 D.15 to each term of the sum. □

Lemma D.18. *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* (C.1.12)).
2. *Suppose that N transfers at least one packet during* $\mathtt{T}$ (*i.e. N sends or receives at least one packet, as on* (C.4.60) *or* (C.4.74–78)). *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. neighbors' signatures*) *pertaining to* $\mathtt{T}$, *OR*

(b) *All of N's signature buffers are clear and N's broadcast buffer* (*or the Data Buffer in the case* $N = S$) *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* (*see footnote* 60) *BB corresponding to some node* $\widehat{N}$*'s testimony 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 testimony 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.*
(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 testimony parcels in N's broadcast buffer correspond to transmission* $\mathtt{T}'$ *and are of the form as indicated on* (C.6.141–144), *where the reason for failure of transmission* $\mathtt{T}'$ *was determined as on* (C.7.190), (C.7.193), *or* (C.7.196).
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 testimony 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 D.10. 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 (A.5.76), (C.4.50), (C.4.80), and (C.4.82). 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 (A.5.89–90), (C.4.50), (C.4.80), and (C.4.82). By the comments on lines (C.4.50), (C.4.53), (C.4.80), and (C.4.82), it is clear that $SIG_{N,N}$ appropriately tracks changes in potential due to the call to *Fill Gap*, while packet movement as on (C.4.53) 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 (A.5.89–90), we see that every packet that is re-shuffled causes a change in $SIG_{N,N}$ of $M - m - 1$ (A.5.76). Notice the actual change in potential matches this amount, since a packet is removed from a buffer at height M (A.5.90), 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 (C.7.191), (C.7.194), and (C.7.197), and they are then cleared at the end of every transmission on (C.7.171) or (C.7.199). 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: (C.4.48–49), (C.4.50), (C.4.74–75), (C.4.80), (C.4.82), (C.6.128), (C.6.133), (C.6.141), (C.6.146), and (A.5.76). The only place that information that was once in one of N's signature buffers is removed from the broadcast buffer is (C.6.134).

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 D.10). For all rounds of all transmissions between $\mathtt{T}+1$ and the time N receives the full *SOT* broadcast for transmission $\mathtt{T}'$, lines (C.4.48–51), (C.4.74–78), and (A.5.76) will never be reached by N (see Lemma D.10 and its proof). Similarly, lines (C.4.80) and (C.4.82) will never be reached since (C.4.63) will always be satisfied.

It remains to consider lines (C.6.128), (C.6.133), (C.6.141), (C.6.146), and (C.6.134); 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 (C.6.128) 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 (C.7.188)). Similar reasoning shows that line (C.6.146) is covered by one of these two cases: If N reaches line (C.6.146) 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 (C.6.142–145) before reaching (C.6.146), 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 (C.6.133–134) 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 (C.6.133–134) in transmission $\widehat{\mathtt{T}}$, N must not have known that $\widehat{N}$ was to be eliminated before that point (C.6.131), and since N received the complete *SOT* broadcast of transmission $\mathtt{T}$ (by Lemma D.10 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 (C.7.188) 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 (C.7.171) (if it was on the blacklist), and as already remarked, $(N, \mathtt{T})$ can never again appear on the blacklist after this.

Now suppose (C.6.141) is reached in some transmission $\widehat{\mathtt{T}} > \mathtt{T}$ and the signature buffers are cleared on this line. Now before line (C.6.141) was reached, by induction on the number of transmissions that have occurred since $\mathtt{T}$, one of the three statements, (2a), (2b), or (2c), was true at the end of transmission $\widehat{\mathtt{T}} - 1$. If (2b) or (2c) was true, then changes made on (C.6.141) will not affect the fact that (2b) or (2c) will remain true. Therefore, assume that we are in Case (2a) before reaching (C.6.141), i.e. that when (C.6.141) is reached in transmission $\widehat{\mathtt{T}}$, N's signature buffers contain the information that they had at the end of $\mathtt{T}$. Since (C.6.141) was reached during $\widehat{\mathtt{T}}$, it must have been that for some $\widetilde{\mathtt{T}}$: $(N, \widetilde{\mathtt{T}}, \widehat{\mathtt{T}})$ was received on (C.6.137) as part of the *SOT* broadcast for transmission $\widehat{\mathtt{T}}$. 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 ((C.3.31–33) and (C.3.35–37)). 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 D.10), either N was not on the blacklist as of the start of transmission $\mathtt{T}$, or N received information as on (C.6.147) indicating N could be removed from the blacklist. Both of these cases imply that by the end of $\mathtt{T}$ and beyond, $(N, \widetilde{\mathtt{T}})$ 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 (2a), if $\widetilde{\mathtt{T}} = \mathtt{T}$, then (C.6.141) 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 D.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 (3a), we track all the times parcels are removed from N's broadcast buffer *BB*, and ensure that if ever N removes a testimony 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' testimony parcels is removed from the sender's data buffer is (C.7.171), 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 (C.6.134), (C.6.139), (C.6.149), (C.6.142–145), and (C.6.154). The former three lines *remove* things from *BB*, while the latter lines *add* things to *BB*. In terms of Statement (3a), we must ensure whenever one of the former three lines is reached, there will never be a testimony 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 (C.6.134), suppose that N reaches line (C.6.134) 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 (C.6.134) requires that N has newly learned that a node has been added to *EN* (C.6.131), let N' denote this node, and let $\mathtt{T}'$ denote the round that N' was eliminated from the network as on (C.7.169–177). First note that necessarily $\mathtt{T}' < \widehat{\mathtt{T}}$. After all, the blacklist will be cleared on line (C.7.171) 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 (C.6.134) 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 testimony parcels for $\widehat{\mathtt{T}}$ (C.6.152).

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 (C.6.137) 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 see that $(N', \mathtt{X})$ will be a part of the *SOT* broadcast for transmission $\mathtt{X}$, indicating that N' has been eliminated (C.7.200). 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, (C.5.110) and (C.5.115))

We next consider when the testimony parcels are removed from *BB* as on (C.6.139). In this case, N has received a *SOT* broadcast parcel of form $(\widehat{N}, \widehat{\mathtt{T}}, \mathtt{T}')$ (C.6.137), and N is removing from *BB* all of $\widehat{N}$'s testimony parcels corresponding to transmissions *other* than $\widehat{\mathtt{T}}$. First note that Lemma D.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 (C.7.188), 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 testimony parcels from $\widehat{N}$ corresponding to $\widehat{\mathtt{T}}$ on (C.6.139), and this is the only transmission for which $\widehat{N}$ can be on the blacklist (at least through $\mathtt{T}'$).

To complete Statement (3a), it remains to consider line (C.6.149). But this is immediate, since if at any time the sender removes $(\widehat{N}, \widehat{\mathtt{T}})$ from the blacklist, then $(\widehat{N}, \widehat{\mathtt{T}})$ can never again be re-added (since nodes are added to the blacklist at the very *end* of a transmission (C.7.188), they are not removed as on (C.6.166) or (C.7.171) 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 (C.7.188) since $\widehat{\mathtt{T}}$ has already passed). Therefore, when N reaches (C.6.149), 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 (C.6.147), 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.

Statement (3a) of the current lemma (now proven) states that no testimony parcel still needed by the sender will ever be deleted from a node's broadcast buffer. Statement (3b) states that a node's broadcast buffer will not hold extraneous testimony parcels, i.e. testimonies 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 (C.6.137), then N will immediately delete all of its testimony parcels from $\widehat{N}$ corresponding to transmissions other than $\mathtt{T}'$ (C.6.139). The fact that the stored parcels have the correct information (i.e. that they address the appropriate reason for failure as on (C.6.142–145)) follows from the fact that

N will only initially store a testimony parcel if it contains the correct information (C.6.153).

4. There are three lines on which the broadcast parcels of the kind relevant to Statement (4) are removed from N's broadcast buffer: (C.6.134), (C.6.139), and (C.6.149). 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 (C.6.134) 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 (C.7.169–177)). If $\widetilde{\mathtt{T}} > \widehat{\mathtt{T}}$, then $(\widehat{N}, \widehat{\mathtt{T}})$ will be cleared from the blacklist on line (C.7.171) 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 (C.7.169–177) and (C.7.188) can never both be reached in a single transmission, see e.g. (C.7.177)). 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 (C.6.137) of transmission $\mathtt{X}$. Clearly, $\mathtt{X} > \widehat{\mathtt{T}}$, since $(\widehat{N}, \widehat{\mathtt{T}})$ can only be added to the blacklist at the end of $\widehat{\mathtt{T}}$ (C.7.188). 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 (C.6.134) 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 (C.6.151). 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 ((C.5.110) and (C.5.115)), we see 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 (C.6.134) 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 (C.6.134) of $\mathtt{T}'$. Therefore, whenever (C.6.134) 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 (C.6.139) or (C.6.149) of some transmission $\mathtt{T}'$. In either case, by looking at the comments on these lines together with Lemma D.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 D.19. *If $A, B \in G$ are honest (not corrupt), in any transmission $\mathtt{T}$ for which both A and B have received the full* SOT *broadcast*:

1. *Between the time B accepts a packet from A on line* (C.4.77) *through the time A gets confirmation of receipt (see Definition* B.8*) for it as on* (C.4.50)*, we have*:
   - $SIG^B[1]_{A,B} = 1 + SIG^A[1]_{A,B}$[61]

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

- $SIG^B[p]_{A,B} = 1 + SIG^A[p]_{A,B}$ (*see footnote* 61)
- $SIG^B[2]_{A,B} = M + SIG^A[3]_{A,B}$, *where* M *is the value of* H_{FP} *on* (C.4.60) (*according to A's view*) *in the same round in which* (C.4.77) *was reached by B*
- $SIG^B[3]_{A,B} = m + SIG^A[2]_{A,B}$, *where* m *is the value of* H_{GP} *on* (C.4.75) (*according to B's view*) *in the same round in which* (C.4.77) *was reached by B*

2. *At all other times, we see 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 (C.2.48) and (C.2.54), and that for any transmission $\mathtt{T}$, both SIG^A and SIG^B are cleared before any packets are transferred (Lemma D.10). 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 (C.4.48–49) and (C.4.74–75), it will be enough to consider only these 4 lines. Furthermore, if lines (C.4.48–49) were reached x times by A in the transmission, and lines (C.4.74–75) were reached y times by B, then:

(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 (C.4.74–75) are necessarily reached *before* lines (C.4.48–49) (i.e. in any transmission, necessarily y will change from zero to 1 *before* x does).

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

We begin by proving Statements (a)–(c). 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 (C.4.48–49)). Also, define y to begin each transmission equal to zero, and to increment by one when line (C.4.74) is reached (just *before* B's signature buffers are updated on lines (C.4.74–75)). Statement (c) is immediate, since RR begins every round equal to -1 (lines (C.2.50) and (C.7.209)), and can only be changed to a higher index on (C.4.78). Therefore, (C.4.46) can never be satisfied before (C.4.78) is reached, which implies (C.4.48) is never reached before (C.4.74) is. We now prove Statement (b). Suppose lines (C.4.48–49) 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 see that B's value for RR (and the one A is using on the comparison on (C.4.45–46)) is at most $\mathtt{t} - 1$. Also, H_{FP} and FR will be set to $\perp$ on (C.4.51) of $\mathtt{t}$. If FR ever changes to a non-$\perp$ value after this, it can only happen on (C.4.56), and so the value it takes must be at least $\mathtt{t}$. Therefore, if at any time after $\mathtt{t}$ we see that $FR \neq \perp$, then if RR has not changed since $\mathtt{t} - 1$, then (C.4.46) can never pass, since $RR \leq \mathtt{t} - 1 < \mathtt{t} \leq FR$. Consequently, (C.4.78) must be reached before

(C.4.48–49) can be reached again after round t, and hence so must (C.4.74–75). This shows that (C.4.48–49) can never be reached twice, without (C.4.74–75) being reached in between.

Conversely, suppose lines (C.4.74–75) are reached in some round t. Notice since we are in round t when this happens, and because *FR* can never have a higher index than the current round index, we see that *A*'s value for *FR* (and the one *B* is using on the comparison on (C.4.73)) is at most t. Also, *RR* will be set to t on (C.4.78) of round t, and *RR* cannot change again until (at some later round) (C.4.73) is satisfied again (or the end of the transmission, in which case there is nothing to show). If line (C.4.56) is NOT reached after (C.4.74–75) of round t, then *FR* can never increase to a larger round index, so *FR* will remain at most t. Consequently, line (C.4.73) can never pass, since if *B* receives the communication from *A* on line (C.4.62), then by the above comments $RR \geq \mathtt{t} \geq FR$. Consequently, (C.4.56) must be reached before (C.4.73) can be reached again after round t. However, by Statement 3 of Claim B.9, (C.4.56) cannot be reached until *A* receives confirmation of receipt from *B* (see Definition B.8), i.e. (C.4.56) can be reached after (C.4.74–75) of round t only if lines (C.4.48–49) are reached.

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

Claim. *Every time line* (C.4.74) *is reached* (*and y is incremented*), *we see that equalities of Statement* 2 *of the lemma are true*, *and between this time and the time line* (C.4.48) *is reached* (*or the end of the transmission*, *whichever comes first*), *we see that the equalities of the first statement of the lemma are true.*

To prove that the claim is true at the outset of any transmission T, notice that before lines (C.4.74–75) 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 claim is true for the base case. We must consider for the induction step two cases: (1) y increases by one (and hence lines (C.4.74–75) will make changes to the signature buffers); and (2) x increases by one (and hence lines (C.4.48–49) have just made changes to the signature buffers). We use the induction hypothesis to assume that the lemma is true at some point, and show that it will continue to be true whenever y and x are incremented by one.

Since (c) guarantees that y gets incremented first, consider a round t in which y is incremented by one (i.e. line (C.4.74) is reached). Since neither x nor y can change between lines (C.3.20) and (C.3.22), by the induction hypothesis we see that the equalities of the second statement of the lemma are true when *A* sends the communication as on (C.4.60) of round t (since y has not been incremented yet at this stage of 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 line (C.4.74), and also *B* updates $SIG^B[3]$ by increasing it by H_{GP} on (C.4.75), we see that the first statement of the lemma will be true after leaving line (C.4.75) (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 (C.4.60) (respectively (C.4.74)) is reached in round t, we will see that immediately after leaving (C.4.75):

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 (C.4.48–49). In the former case, the Claim certainly remains true. In the latter case, let $\mathtt{t}'$ denote the time that (C.4.48) is next reached. Notice that $\mathtt{t}' > \mathtt{t}$, as Statement (b) above guarantees (C.4.48) is reached *after* (C.4.74), and by examining the pseudo-code, this cannot happen until at least the next round after $\mathtt{t}$. Also, the fact that (C.4.48) is reached in round $\mathtt{t}'$ implies that (C.4.45) was satisfied, and in particular, A must have received the communication from B as on (A.3.07) of round $\mathtt{t}'$. And since $\mathtt{t}' > \mathtt{t}$, the values received on (C.3.07) 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 (C.4.74–75) when B sends A the communication on (C.3.11)). 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 (C.4.49), that the new value for $SIG^A[3]$ equals the value stored in $SIG^B[2]$. Since before (C.4.49) 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 (C.4.49), that the value of H_{FP} there equals M. We argue that this by showing H_{FP} will not change from line (C.4.60) of round $\mathtt{t}$ (when M was set to H_{FP}) through line (C.4.49) of round $\mathtt{t}'$. To see this, notice that the only possible places H_{FP} can change *during* a transmission are lines (C.4.51), (C.4.53), and (C.4.56). Clearly, (C.4.51) cannot be reached between these times, since (C.4.49) is not reached during these times. Also, Statement 3 of Claim B.9 implies that (C.4.56) cannot be reached between these times either. Finally, (C.4.53) cannot be reached, since RR will be set to $\mathtt{t}$ on (C.4.78) of round $\mathtt{t}$, and by statement (b), (C.4.78) cannot be reached again until after (C.4.49) is reached in round $\mathtt{t}'$, and hence RR will be equal to $\mathtt{t}$ from (C.4.78) of round $\mathtt{t}$ through (C.4.49) of round $\mathtt{t}'$. Also, FR will not change between these times (also by Statement 3 of Claim B.9), and since the only non-$\perp$ value FR is ever set to is the current round as on (C.4.56), we see that $FR \leq \mathtt{t}$. Putting these facts together, we see that for all times between line (C.4.60) of round $\mathtt{t}$ through line (C.4.49) of round $\mathtt{t}'$, either A does not receive RR (in which case $RR = \perp$ when (C.4.52) is reached) or A receives RR, which as noted obeys $RR = \mathtt{t} \geq FR$. In either case, (C.4.52) will fail, and (C.4.53) cannot be reached. □

Lemma D.20. *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* (C.7.187). *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 does not get 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 (C.7.187) and for any honest node $A \in G$, then along (directed) edge $E(A, N)$, A

will never reach line (C.4.60), and along (directed) edge $E(N, A)$, A will never reach lines (C.4.67–82). Fix a transmission $\mathtt{T}$ in which (C.7.187) is reached (i.e. a node is not eliminated as on (C.7.169–177) of $\mathtt{T}$), let $N \notin \mathcal{P}_{\mathtt{T}}$ be any node *not* put on $\mathcal{P}_{\mathtt{T}}$ on (C.7.187) of $\mathtt{T}$, and let $A \in G$ be an honest node. Since $N \notin \mathcal{P}_{\mathtt{T}}$, we see that either $N \in EN$ or $N \in BL$ when (C.7.187) is reached. Since no nodes can be *added to EN* or *BL* from the outset of $\mathtt{T}$ through line (C.7.187) of $\mathtt{T}$, we must have $N \in EN$ or $N \in BL$ as of either line (C.7.188) or (C.7.170) of a previous transmission. Therefore, either $(N, \mathtt{T})$ or $(N, \mathtt{T}', \mathtt{T})$ is added to the *SOT* broadcast of transmission $\mathtt{T}$ (on (C.7.176) or (C.7.200) 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 D.10. 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 (C.4.59), (C.3.31–33), (C.4.63) and (C.3.35–37), A will not transfer any packets with N. □

Lemma D.21. *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* (C.3.29).

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 (C.3.28) and (C.7.178–179)), 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 $\mathsf{P}_{\mathtt{t}} = N_0 N_1 \dots N_L$ (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 (C.3.15). Looking at the manner in which broadcast parcels are chosen (C.5.115), 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 D.22. *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 D.23. *For every transmission* $\mathtt{T}$ *for which line* (C.7.187) *is reached*: $S, R \in \mathcal{P}_{\mathtt{T}}$.

Proof. The participating list $\mathcal{P}_{\mathtt{T}}$ is set on line (C.7.187) at the end of every transmission (except transmissions for which a node is eliminated as on (C.7.169–177)). By looking at the code there, we must show that $S, R \notin EN \cup BL$ at the end of any transmission for which line (C.7.187) is reached. That an honest node can never be identified as corrupt and eliminated is the content of the proof of Theorem 5.2, so $S, R \notin EN$. Since S is never put on the blacklist (C.7.188), it remains to show $R \notin BL$ when (C.7.187) is reached. Since nodes are removed from the blacklist on line (C.6.166) and not put on it again until (C.7.188), 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 (C.6.166) of transmission $\mathtt{T}$. If R is ever placed on the blacklist, we argue that: (1) R will learn what testimony 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 $\mathtt{T}-1$ (C.7.203). 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 (C.7.178–179). Therefore, for the first n^3 rounds, the sender's *SOT* broadcast will have top priority in terms of sending/receiving broadcast parcels (C.5.115). Since S and R are connected by an active honest path at each round, we follow the proof as in Lemma D.21 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 D.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 testimony) to its broadcast buffer (C.6.137–145). Looking at the proof of Lemma D.31 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 (C.5.115), and since the receiver is connected to the sender *every* round via some active honest path (by the conforming assumption), its requested testimony information will necessarily reach the sender within $4n^3$ rounds, proving (2). □

Lemma D.24. *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 (C.7.187). Since the only place the sender adds nodes to the blacklist is on (C.7.188), 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 (C.7.187) 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 (C.4.59) will always fail for any other node, see (C.3.31–33)). Lemma D.31 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 D.20), 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 Slide protocol by Lemma D.1. In particular, the transmission will necessarily be successful. □

Lemma D.25. *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 (C.5.110) 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 (C.7.188). Therefore, after receiving the information for the first time that $\widehat{N}$ should be removed, the comments on line (C.6.123) will guarantee A will not accept additional blacklist information regarding $\widehat{N}$ until the following transmission, proving the lemma. □

Lemma D.26. *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 testimony.*

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 (C.6.123)). Therefore, as long as the parcel is not deleted from the broadcast buffer, line (C.5.115) 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 testimony) are only removed on (C.6.139) and (C.6.149), we need only consider the case that (C.6.149) 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 (C.6.149) deletes from N's broadcast buffer the parcel indicating that N knows $\widehat{N}$'s complete testimony, then N will never again add a parcel of this form to its broadcast buffer (as on (C.6.155)) for the remainder of $\mathtt{T}$. But this is immediate, since if N removes this parcel from BB on (C.6.149) of $\mathtt{T}$, then $\widehat{N}$ must have been removed from the blacklist (see (C.6.147)), and since $\widehat{N}$ cannot be re-added to the blacklist until the end of $\mathtt{T}$ (C.7.188), line (C.6.152) (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 (C.6.155) cannot be reached. This proves that once the parcel is deleted, it cannot be later added in the same transmission, proving the lemma. □

Lemma D.27. *If there is a transmission* $\mathtt{T}$ *for which the sender has collected the complete testimonies from every node that participated in* $\mathtt{T}$*, then the sender can necessarily identify a corrupt node.*

Proof. Each failed transmission falls under F2, F3, or F4, and the lemma is proven for each case below in Theorems D.28, D.33 and D.34. □

D.1. *Handling Failures as in F2: Packet Duplication*

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

Theorem D.28. *Suppose transmission* $\mathtt{T}$ *failed and falls under case F*2*, and at some later time* (*after transmission* $\mathtt{T}$ *but before any additional nodes have been eliminated*) *the sender has received all of the testimony 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 4.7), 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 information contained in the testimonies, 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 proof of this fact will require some work. We begin with the following definitions:

Definition D.29. 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 D.30. 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* (C.3.31) or *Okay To Receive Packet* (C.3.35) 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 D.31. *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 testimony for a blacklisted node*
4. *A node on* $\mathsf{P}_\mathtt{t}$ *learns a testimony 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 testimony 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* (C.3.31) and (C.3.35), we first argue 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 (C.3.15). 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 (C.5.115)) 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 (C.7.200), then find the last index $0 \leq j < i$ 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 (C.5.115) 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 testimony 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 (C.5.115).
- 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 (C.5.115).

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 $0 \leq j < i + 1$ 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 (C.7.203) and restored when they receive the *SOT* broadcast (C.7.200), (C.6.137–138)). 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 (C.6.147–149). 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 (C.5.115) 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 exists some $0 \leq j < i$ such that N_j does *not* have N_i's complete testimony, but N_{j+1} does (since $N_i \in BL_S$ implies S does not have the complete testimony, but N_i has its own complete testimony in its broadcast buffer, see Statement 2 of Lemma D.18). 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 D.18 implies that N_j is aware that N_{j+1} knows the complete testimony of N_i (who by choice of j is on N_j's blacklist), and hence α will necessarily be set as on (C.5.119 or C.5.122) and sent to N_j on (C.5.103)). Consequently, N_{j+1} will receive α (C.5.105) during Stage 1 communication of round $\mathtt{t}$, and will have broadcast information of type 5 (C.5.115) 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 $\mathtt{t}$ (C.3.15), 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 (C.7.179), and the *SOT* is at most $2n - 1$ parcels (C.7.200), 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 see that Case 1 can happen at most n^3 times.
2. Lemma D.25 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 D.26 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 testimony. 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 testimony parcel. Since there are at most $n-1$ blacklisted nodes at any time (see (C.7.187–188) and Claim D.6), and at most n testimony parcels per blacklisted node (see (C.6.142–45) and Lemma D.7), an honest node can *learn a new* testimony parcel at most $n(n-1) < n^2$ times per transmission (see Statement 3 of Lemma D.18 which says honest nodes will not ever "unlearn" relevant testimony 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 Cases 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 testimonies contain in the case of F2 (see (C.1.12), (C.1.17), (C.1.32), and (C.6.142–145)):[62]

- $SIG_{A,A}$ denotes the net decrease in A's potential due to re-shuffling packets in the current transmission.
- $SIG^A[2]_{A,B}$ 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 ((C.4.75), (C.3.11), and (C.3.07)).
- $SIG^A[2]_{B,A}$ 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 (C.4.60), (C.4.62), and (C.4.74).
- $SIG^A[3]_{A,B}$ 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 (C.4.49).
- $SIG^A[3]_{B,A}$ 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 (C.4.75).

Lemma D.32. *Suppose transmission* $\mathtt{T}$ *failed and falls under case F*2, *and at some later time* (*after transmission* $\mathtt{T}$ *but before any additional nodes have been eliminated*) *the sender has received all of the testimonies from every node on* $\mathcal{P}_{\mathtt{T}}$. *Then one of the following two things happens*:

[62] 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 (C.3.11) and (C.4.59–60)).

1. *There is some node $A \in G$ whose testimony indicates that A is corrupt.*[63]
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*

$$4n^3 - 4n^2 + \sum_{B \in \mathcal{P} \setminus A} SIG^A[3]_{B,A} < SIG_{A,A} + \sum_{B \in \mathcal{P} \setminus A} SIG^B[2]_{A,B}, \qquad \text{(D.31)}$$

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 (each of its $2(n-1)$ buffers have height $2n$, and hence each have maximum capacity: $\sum_{i=1}^{2n} i = n(2n+1)$).

Proof. The idea of the proof is to use Lemma D.14, 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 $3D$ 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 an upper bounded from the capacity of the network) is $4n^4$ (each of the $n-2$ internal nodes can be responsible for at most $4n^3$ to potential, see e.g. the last sentence of the lemma) 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,

[63] 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 testimony. By "outdated" information, we mean that as part of its testimony, 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 testimony, 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.

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.*

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

Proof. The above inequalities state that for honest nodes, the potential changes due to re-shuffling and packet transfers are strictly non-positive (this was the content of Lemma 4.11). This observation is proved as Statements 4 and 5 of Lemma D.11 in Appendix D. □

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*:

$$\sum_{A \in \mathcal{P} \setminus S} SIG^A[3]_{S,A} < 2nD + 2n^2. \tag{D.32}$$

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 (D.32) 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 B.8). 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 (D.32). The proof can be found in Lemma D.17 in Appendix D. □

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 $4D$ (C.3.02), we see 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*:

$$2nD + 4n^4 - 4n^3 + 2n^2 < \sum_{A\in\mathcal{H}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{H}\setminus S}\sum_{B\in\mathcal{P}\setminus\{A,S\}} \left(SIG^A[2]_{B,A} - SIG^A[3]_{B,A}\right). \quad \text{(D.33)}$$

Proof. This follows immediately from Observation 3 and Lemma D.14, since:

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

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*

$$4n^3 - 4n^2 < SIG_{A,A} + \sum_{B\in\mathcal{P}\setminus A} SIG^B[2]_{A,B} - SIG^A[3]_{B,A}. \quad \text{(D.34)}$$

Proof. Consider the following inequalities:

$$\begin{aligned} &2nD + 4n^4 - 4n^3 + 2n^2 \\ &< \sum_{A\in\mathcal{H}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{H}\setminus S}\sum_{B\in\mathcal{P}\setminus\{A,S\}} \left(SIG^A[2]_{B,A} - SIG^A[3]_{B,A}\right) \\ &\leq \sum_{A\in\mathcal{P}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{P}\setminus S}\sum_{B\in\mathcal{P}\setminus\{A,S\}} \left(SIG^A[2]_{B,A} - SIG^A[3]_{B,A}\right) \\ &= \sum_{A\in\mathcal{P}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{P}\setminus S}\sum_{B\in\mathcal{P}\setminus\{A,S\}} \left(SIG^B[2]_{A,B} - SIG^A[3]_{B,A}\right). \quad \text{(D.35)} \end{aligned}$$

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-labeling of terms. Subtracting

$2nD + 2n^2$ from both sides:

$$\begin{aligned}
4n^4 - 4n^3 &< \sum_{A\in\mathcal{P}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{P}\setminus S}\sum_{B\in\mathcal{P}\setminus\{A,S\}} \left(SIG^B[2]_{A,B} - SIG^A[3]_{B,A}\right) \\
&\quad - 2nD - 2n^2 \\
&< \sum_{A\in\mathcal{P}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{P}\setminus S}\sum_{B\in\mathcal{P}\setminus\{A,S\}} \left(SIG^B[2]_{A,B} - SIG^A[3]_{B,A}\right) \\
&\quad + \sum_{A\in\mathcal{P}\setminus S} -SIG^A[3]_{S,A} \\
&= \sum_{A\in\mathcal{P}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{P}\setminus S}\sum_{B\in\mathcal{P}\setminus\{A,S\}} \left(SIG^B[2]_{A,B} - SIG^A[3]_{B,A}\right) \\
&\quad + \sum_{A\in\mathcal{P}\setminus S} \left(SIG^S[2]_{A,S} - SIG^A[3]_{S,A}\right) \\
&= \sum_{A\in\mathcal{P}\setminus S} SIG_{A,A} + \sum_{A\in\mathcal{P}\setminus S}\sum_{B\in\mathcal{P}\setminus A} \left(SIG^B[2]_{A,B} - SIG^A[3]_{B,A}\right). \quad \text{(D.36)}
\end{aligned}$$

Above, the top inequality is from (D.35), 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 (C.3.21–22)), 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

$$4n^3 - 4n^2 < SIG_{A,A} + \sum_{B\in\mathcal{P}\setminus A} \left(SIG^B[2]_{A,B} - SIG^A[3]_{B,A}\right), \quad \text{(D.37)}$$

which is (D.34). □

Therefore, if a node cannot be eliminated as in Obs. 1 or 2 (which are covered by Case 1 of Lemma D.32), then Obs. 5 implies that Case 2 of Lemma D.32 is true. □

Proof of Theorem D.28. This Theorem now follows immediately from Lemma D.32 and the fact that a node $A \in G$ for which (D.31) 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 D.16. □

D.2. *Handling Failures as in F3: Packet Deletion*

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

Theorem D.33. *Suppose transmission* $\mathtt{T}$ *failed and falls under case F*3, *and at some later time* (*after transmission* $\mathtt{T}$ *but before any additional nodes have been eliminated*) *the sender has received all of the testimony 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 testimonies 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 testimonies contain in the case of F3 ((C.1.17), (C.1.32), and (C.6.144)):

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

By the third Statement of Lemma D.15, either a corrupt node can be eliminated, or the following is true for all $A, B \in G$:

$$\left|SIG^A[1]_{B,A} - SIG^B[1]_{B,A}\right| \leq 1 \quad \text{and} \quad \left|SIG^A[1]_{A,B} - SIG^B[1]_{A,B}\right| \leq 1.$$

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

$$\sum_{A,B\in\mathcal{P},A\neq B} \left|SIG^A[1]_{B,A} - SIG^B[1]_{B,A}\right| \leq n^2. \tag{D.38}$$

This in turn implies that

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

The first inequality is from (D.38), the second line is from re-labeling 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 (C.3.16–20) and (C.3.21–22)), 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.[64] Using an averaging argument, we can find some $A \in G$ such that

$$4n^2 - 8n < 6n^2 - n < \sum_{B \in \mathcal{P} \setminus A} \left(SIG^A[1]_{B,A} - SIG^A[1]_{A,B} \right), \tag{D.39}$$

where the first inequality is obvious. Statement 7 of Lemma D.11 now guarantees that A is corrupt.[65] □

D.3. *Handling Failures as in F4: Packet Duplication + Deletion*

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

Theorem D.34. *Suppose transmission* $\mathtt{T}$ *failed and falls under case F*4, *and at some later time* (*after transmission* $\mathtt{T}$ *but before any additional nodes have been eliminated*) *the sender has received all of the testimony 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 testimonies, 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.

Proof. By definition of F4, the receiver received some packet p (corresponding to the current codeword) at least twice. Therefore, when (C.7.178–179) 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 D.21), and this is in turn broadcasted as part of the sender's *start of transmission* broadcast in the following transmission ((C.7.190–192) and (C.7.200)). We will use the following notation to describe the specific information the testimonies contain in the case of F4 (see (C.1.17) and (C.1.32)):

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

[64] 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.

[65] Intuitively, A must be corrupt since the sum on the RHS of (D.39) 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.

Consider the following string of equalities:

$$\begin{aligned}
0 &= \sum_{A\in\mathcal{P}_T}\sum_{B\in\mathcal{P}_T}\left(SIG^A[p]_{B,A} - SIG^A[p]_{B,A}\right)\\
&= \sum_{A\in\mathcal{P}_T}\sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}\right)\\
&= \sum_{A\in\mathcal{P}_T\setminus\{R,S\}}\sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}\right)\\
&\quad + \sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{R,B} - SIG^R[p]_{B,R}\right)\\
&\quad + \sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{S,B} - SIG^S[p]_{B,S}\right)\\
&= \sum_{A\in\mathcal{P}_T\setminus\{R,S\}}\sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}\right)\\
&\quad + \sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{S,B} - SIG^R[p]_{B,R}\right). \qquad \text{(D.40)}
\end{aligned}$$

The first equality is trivial, the second equality comes from re-labeling and re-arranging 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 (C.3.16–20) and (C.3.21–22)). 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,[66] we have

$$\sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{S,B} - SIG^R[p]_{B,R}\right) \le -1. \qquad \text{(D.41)}$$

Plugging this into (D.40) and re-arranging:

$$1 \le \sum_{A\in\mathcal{P}_T\setminus\{R,S\}}\sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}\right). \qquad \text{(D.42)}$$

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

$$1 \le \sum_{B\in\mathcal{P}_T}\left(SIG^B[p]_{A,B} - SIG^A[p]_{B,A}\right). \qquad \text{(D.43)}$$

Now Statement 8 of Lemma D.11 says that A is necessarily corrupt.[67] □

[66] This was proven in Observations 2–3 of Lemma B.15 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).

[67] Intuitively, A is corrupt since (D.43) says that it has output p more times than it input p.

References

[1] Y. Afek, E. Gafni, End-to-end communication in unreliable networks, in *Proc. of the 7th ACM Symp. on Principles of Distributed Computing* (1988), pp. 131–148

[2] Y. Afek, E. Gafni, A. Rosén, The slide mechanism with applications in dynamic networks, in *Proc. of the 11th ACM Symp. on Principles of Distributed Computing* (1992), pp. 35–46

[3] Y. Afek, B. Awebuch, E. Gafni, Y. Mansour, A. Rosen, N. Shavit, *Slide*—the key to polynomial end-to-end communication. *J. Algorithms* **22**, 158–186 (1997)

[4] W. Aiello, E. Kushilevitz, R. Ostrovsky, A. Rosén, Adaptive packet routing for bursty adversarial traffic. *J. Comput. Syst. Sci.* **60**(3), 482–509 (2000)

[5] B. Awerbuch, T. Leighton, Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks, in *Proc. 26th ACM Symp. on Theory of Computing* (1994), pp. 487–496

[6] B. Awerbuch, Y. Mansour, N. Shavit, End-to-end communication with polynomial overhead, in *Proc. of the 30th IEEE Symp. on Foundations of Computer Science, FOCS* (1989)

[7] B. Awerbuch, D. Holmer, C. Nina-Rotaru, H. Rubens, An on-demand secure routing protocol resilient to byzantine failures, in *Proc. of 2002 Workshop on Wireless Security* (2002), pp. 21–30

[8] B. Barak, S. Goldberg, D. Xiao, Protocols and lower bounds for failure localization in the Internet, in *Proc. of Advances in Cryptology—27th EUROCRYPT 2008*. LNCS, vol. 4965 (Springer, Berlin, 2008), pp. 341–360

[9] P. Bunn, R. Ostrovsky, Asynchronous throughput-optimal routing in malicious networks, in *Proc. 37th International Colloquium on Automata, Languages, and Programming*. LNCS, vol. 6199 (Springer, Berlin, 2010), pp. 236–248

[10] P. Bunn, R. Ostrovsky, Secure end-to-end communication with optimal throughput in unreliable networks (2013). arXiv:1304.2454

[11] S. Even, O. Goldreich, S. Micali, On-line/off-line digital signatures. *J. Cryptol.* **9**(1), 35–67 (1996)

[12] S. Goldberg, D. Xiao, E. Tromer, B. Barak, J. Rexford, Path-quality monitoring in the presence of adversaries. *ACM SIGMETRICS* **36**, 193–204 (2008)

[13] O. Goldreich, *The Foundations of Cryptography, Basic Applications* (Cambridge University Press, Cambridge, 2004)

[14] E. Kushilevitz, R. Ostrovsky, A. Rosén, Log-space polynomial end-to-end communication. *SIAM J. Comput.* **27**(6), 1531–1549 (1998)

[15] S. Micali, C. Peikert, M. Sudan, D. Wilson, Optimal error correction against computationally bounded noise, in *Proc. of 2nd Theory of Cryptography Conf.* LNCS, vol. 3378 (Springer, Berlin, 2005), pp. 1–16

[16] R. Perlmann, Network layer protocols with byzantine robustness. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1988)

[17] S. Rajagopalan, L. Schulman, A coding theorem for distributed computation, in *Proc. 26th ACM Symp. on Theory of Computing* (1994), pp. 790–799

[18] L. Schulman, Coding for interactive communication. *IEEE Trans. Inf. Theory* **42**(6), 1745–1756 (1996). Special issue on Codes and Complexity, Part I. (Preliminary versions: Proc. 33rd FOCS 724–733, 1992 and Proc. 25th STOC 747–756, 1993)

[19] A. Shamir, Y. Tauman, Improved online/offline signature schemes, in *Proc. of 21st Advances in Cryptology, CRYPTO 2001*. LNCS, vol. 2139 (Springer, Berlin, 2001), pp. 355–367