New Dolev-Reischuk Lower Bounds Meet Blockchain Eclipse Attacks

ITTAI ABRAHAM, VMWare Research, Israel

GILAD STERN, The Hebrew University in Jerusalem, Israel

In 1985, Dolev and Reischuk proved a fundamental communication lower bounds on protocols achieving fault tolerant synchronous broadcast and consensus: any deterministic protocol solving those tasks requires at least a quadratic number of message to be sent by nonfaulty parties. Followup work by Abraham, Chun, Dolev, Nayak, Pass, Ren and Shi shows a similar lower bound for randomized protocols. With the rise of blockchain systems, there have been many real-world systems that achieve consensus with seemingly linear communication per instance. We bridge this discrepancy in two ways. First, we generalize the lower bound to Crusader Broadcast protocols, and to all-but m Crusader Broadcast. Second, we discuss the ways these lower bounds relate to the security of blockchain systems. Specifically, we show how *eclipse style attacks* in such systems can be viewed as specific instances of *Dolev-Reischuk style attacks*. Our observation suggests a more systematic way of analyzing and thinking about eclipse style attacks through the lens of the Dolev-Reischuk family of attacks. Finally, we present an example of a simple subquadratic Crusader Broadcast protocol whose security is highly dependent on insights from the presented lower bounds.

1 INTRODUCTION

Two of the foundational and highly related tasks in the world of distributed systems are *consensus*, and *broadcast*. In a consensus protocol, all parties have some input and they must agree on an output. On the other hand, in a broadcast protocol a designated sender attempts to send a specific message to all parties, and all parties must output the same message sent by the sender. These tasks have been widely researched both in theoretic settings and in practical settings. Ideally we would like to be able to design efficient protocols for solving these tasks in the presence of faults. A foundational limit on the efficiency of such protocols has been shown in the work of Dolev and Reischuk in 1985 [7]. They prove that any deterministic protocol solving fault tolerant broadcast must send at least $\Omega(n \cdot f)$ messages, where n is the number parties overall and f is the number of omission-faulty parties, whose incoming and outgoing messages can be dropped. Since broadcast and consensus reduce to each other [5], the lower bound also provides a lower bound on consensus. A work by Abraham *et al.* [1] then generalized this work to probabilistic protocols, showing that with f *Byzantine* faults, a broadcast protocol with a $\frac{3}{4} + \epsilon$ probability of success requires $\Omega(\epsilon nf)$ messages to be sent in expectation.

Recent years have seen growing interest in blockchain protocols and systems. In essence, these systems are designed to solve the task of consensus [16, 19], or more precisely, state machine replication. Many of these systems actually achieve consensus in a **linear** number of messages per agreed upon value. This fact seems to be in direct conflict with the lower bounds of Dolev-Reischuk and Abraham *et al.*, suggesting that at least **quadratic** ($\Omega(nf)$) messages are required. One way we could try to make sense of this contradiction is by looking at the details of the lower bounds. Dolev-Reischuk prove such lower bounds for protocols in which parties are required to output some value eventually, even without hearing any message. In real-world systems however, it is entirely reasonable not to make a decision unilaterally without hearing about it. In this work we think of that as outputting a special value $\perp$, signifying not knowing.

We explore this gap between theory and practice even further. First, we show that, perhaps surprisingly, similar lower bounds exist even for the task of *Crusader Broadcast*, in which parties are allowed to remain undecided when the sending is fualty. This is done by showing an attack on such protocol, executed by a strongly adaptive adversary that

Authors' addresses: Ittai Abraham, VMWare Research, Herzliya, Israel; Gilad Stern, The Hebrew University in Jerusalem, Jerusalem, Israel.

can simulate other parties. We then explore the connections between the attack style used in our lower bounds and the eclipse style attacks in proof-of-work blockchain systems. We show that in practice, many of these attacks can in fact be executed by a highly static adversary, not requiring strong adaptivity. In addition, one would think that an adversary cannot simulate other parties in proof-of-work systems with virtually unbreakable cryptography. However, we show that some of the eclipse attacks suggested in the literature exactly rely on the ability to simulate certain behaviour of other nodes. This can be done by dropping all of their communication, or even worse, by filtering some information to them and making them "simulate themselves". Finally, we show that when the adversary is not adaptive and cannot simulate, then in fact, subquadratic is possible. We provide a simple subquadratic protocol, which is secure as long as the adversary is static and cannot simulate other parties, but is not secure if that is not the case.

In more detail, our work makes three contributions:

(1) From a theory and foundations point of view, we extend the Dolev-Reischuk type attacks (lower bound) by showing in Theorem 1 that similar attacks holds for easier tasks. While Dolev-Reischuk shows that broadcast (and consensus) needs $\Omega(nf)$ messages, we extend this to show that even the weaker *Crusader Broadcast* (and *Crusader Consensus*) needs $\Omega(nf)$ messages, with f being the number of Byzantine parties. In a Crusader Broadcast protocol, if the sender is nonfaulty, all parties output the message it sent. However, if the sender is faulty, there exists some value v such that all nonfaulty parties either output v or $\perp$, making the task strictly easier than broadcast. As most of these protocols are designed to be secure in the face of a constant fraction of Byzantine parties, this implies that at least $\Omega(n^2)$ messages need to be sent in Crusader Broadcast protocols as well. Similar to how Abraham *et al.* [1] extend the Dolev-Reischuk lower bound to the randomized setting given a strongly adaptive adversary, we also extended our lower bound on Crusader Broadcast to the randomized setting given a strongly adaptive adversary in Theorem 2. We generalize the lower bound even further to the task of all-but m Crusader Broadcast. In this task all parties must output the same value v or $\perp$, except for m parties which are each allowed to output any $v' \notin \{v, \perp\}$. Theorem 3 shows that an all-but $(f^c - 1)$ protocol requires $\Omega(nf^{1-c})$ messages to be sent for any $c \in [0, 1]$.
The extension of Dolev-Reischuk from broadcast to Crusader Broadcast is non-trivial and requires more from the adversary. In particular all the adversary needs to do in the classic Dolev-Reischuk is to omit messages. In fact it has been observed that Dolev-Reischuk holds even for omission failures [3]. In contrast, for Crusader Broadcast, the adversary needs to act in a Byzantine manner. Moreover, similar to the classic lower bound of Fischer, Lynch, and Merritt [10], for our lower bounds to hold, the adversary needs to be able to simulate a non-trivial number of non-faulty parties.

(2) Our second contribution is making the conceptual connection between the classic Dolev-Reischuk type attacks on protocols with $o(nf)$ messages and the practical eclipse type attacks on blockchain systems.
Concretely we show that eclipse type attacks can be viewed as attacks on specific Crusader Broadcast protocols that use $o(nf)$ messages. We discuss the similarities between isolating parties in real-world eclipse attacks and in the lower bounds presented in this paper. We then discuss how performing a double spend attack, a 51% percent attack, and selfish mining attacks are specific cases of the attacks presented in our lower bounds. Note that attacks that require an adaptive adversary, as formulated in our lower bounds, are extremely hard to execute in the real world. Such attacks would require an adversary to to corrupt parties on the fly, using information it sees throughout the run to choose which parties to corrupt. A strongly adaptive attack is even harder to execute, requiring an adversary to intercept messages before they are delivered, corrupt the sending

parties, and replace the messages with different ones. Unfortunately, many of the real-world eclipse attacks could be executed even by a highly static adversary which preemptively influences the communication network. This suggests that some blockchain systems did not force the adversary to be "as adaptive" as it needs to be in order to execute the strongest Dolev-Reischuk style attacks.

(3) Building on the two previous contributions, we suggest a new and more principled method of defending against eclipse attacks. Instead of patching each eclipse attack as they appear, we suggest that in practice, for $o(n^2)$ type protocols to be safe they should be built in a way that either (1) forces the adversary to be highly adaptive; or (2) to actively simulate a large portion of the network, or both. Said differently, any suggested protocol with $o(n^2)$ messages should be built in such a way that it does not succumb to a Dolev-Reischuk type attack of a highly static non-simulating adversary. For completeness, we show a specific $o(n^2)$ algorithm and prove that it is secure against a static non-simulating adversary. This protocol is used as an example because it is simple and straight forward to understand. It can be improved upon with techniques used in randomized broadcast protocols.

2 COMMUNICATION AND ADVERSARY MODEL

In this work we consider a synchronous fully-connected network of n parties. In a synchronous network, there is a commonly known bound Δ on message delay. This means that if a party sends a message at time t, it is guaranteed to be delivered by time $t + \Delta$. The adversary can choose exactly how long each message is delayed within the range $[0, \Delta]$. The lower bounds presented in this work deal with an even stronger synchrony assumption: lockstep communication. In such a setting communication proceeds in rounds. Parties can send messages in the beginning of each round, which are then delivered by the end of the round. Note that this actually limits the power of the adversary, seeing as it could choose to allow lockstep communication, and thus strengthens the lower bound. For example, an adversary could choose to make communication proceed in rounds by delivering all pending message at intervals of Δ time, i.e. at time $i \cdot \Delta$ for every $i \in \mathbb{N}$.

This work deals with two different types of Byzantine adversary: a static Byzantine adversary and a strongly adaptive Byzantine adversary. In both cases, the adversary can corrupt up to f parties, causing them to arbitrarily deviate from the protocol. A static Byzantine adversary must choose which parties to corrupt in the beginning of the protocol. On the other hand, a strongly adaptive adversary can choose which parties to corrupt at any given time. Furthermore, it can even choose to corrupt parties after they send messages, but before they are delivered. If it chooses to do so, it can delete those message and send different messages instead. Regardless of the type of adversary, in this work we assume the adversary can simulate other parties if required.

3 DEFINITIONS

The Binary Crusader Broadcast task is very similar to the Binary Broadcast task, except parties are also allowed to output $\perp$ if the sender is faulty. Formally, such a protocol is defined as follows:

Definition 1. *A Binary Crusader Broadcast protocol has a designated sender s with some input $x \in \{0, 1\}$. Every party outputs some value $y_i \in \{0, 1, \perp\}$. A protocol solving Binary Crusader Broadcast has the following properties:*

- ***Validity.*** *If the sender is nonfaulty, then every nonfaulty party outputs x.*
- ***Correctness.*** *If two nonfaulty parties $i \neq j$ output $y_i, y_j \in \{0, 1\}$, then $y_i = y_j$.*
- ***Termination.*** *If all nonfaulty parties participate in the protocol, they all complete it.*

A weaker version of the Binary Crusader Broadcast task is the almost-everywhere Binary Crusader Broadcast protocol, similar to the almost-everywhere agreement problem [8, 18]. Whereas in a regular Binary Crusader Broadcast protocol all parties that don't output $\perp$ must output the same value even when the sender is faulty, in an almost-everywhere Binary Crusader Broadcast protocol a small number of parties are allowed to output a different value when the sender is faulty. In a Binary Crusader Broadcast protocol, this is equivalent to saying that either the number of parties that output 0 or the number of parties that output 1 must be small. A protocol solving Binary Crusader protocol allowing m parties to disagree is called an all-but m Binary Crusader protocol, and is defined as follows:

Definition 2. *An all-but m Binary Crusader Broadcast protocol has a designated sender s with some input $x \in \{0, 1\}$. Every party outputs some value in $y_i \in \{0, 1, \perp\}$. A protocol solving all-but m Binary Crusader Broadcast has the following properties:*

- ***Validity.*** *If the sender is nonfaulty, then every nonfaulty party outputs x.*
- ***Correctness.*** *Either at most m nonfaulty parties output* 0 *or at most m nonfaulty parties output* 1.
- ***Termination.*** *If all nonfaulty parties participate in the protocol, they all complete it.*

A protocol is said to be deterministic if all nonfaulty parties' actions are chosen as a deterministic function of their input and the messages they receive, and probabilistic otherwise. A protocol is said to be p-correct if for any adversary all of its properties hold with probability p or greater.

3.1 Relationship to Crusader Consensus

The notion of Crusader Broadcast is highly related to that of Crusader Consensus. We define the task of Crusader Consensus as follows:

Definition 3. *In a Binary Crusader Consensus protocol, every party i has an input $x_i \in \{0, 1\}$. Every party outputs some value $y_i \in \{0, 1, \perp\}$. A protocol solving Binary Crusader Consensus has the following properties:*

- ***Validity.*** *If all nonfaulty parties have the same input x, then they all output x.*
- ***Correctness.*** *If two nonfaulty parties $i \neq j$ output $y_i, y_j \in \{0, 1\}$, then $y_i = y_j$.*
- ***Termination.*** *If all nonfaulty parties participate in the protocol, they all complete it.*

These tasks reduce to each other in the same way regular consensus and broadcast reduce to each other when $n \geq 2f + 1$ with f Byzantine parties [5]. In short, assume we have a Crusader Broadcast protocol. In order to achieve Crusader Consensus, every party broadcasts its input x_i and waits to complete all broadcasts. After completing all n broadcasts, if there exists some value y which was received in at least $n - f$ broadcasts, output y. Otherwise output $\perp$. If all nonfaulty parties have the same input x, then from the Validity property they will receive that value in at least the $n - f$ broadcasts with nonfaulty senders and output it. On the other hand, if some nonfaulty party outputs a value $y \neq \perp$, then it received it in at least $n - f$ broadcasts. From the Correctness property, every other party either outputs y or $\perp$ in those broadcasts, and thus can output some y' such that $y' \notin \{y, \perp\}$ only in the f remaining broadcasts. We know that $n \geq 2f + 1$, and thus $f \leq n - (f + 1) < n - f$, which means that it won't output $y' \notin \{y, \perp\}$, as required.

In the other direction, assume that there exists a Crusader Consensus protocol. In order to implement broadcast, the sender sends its input to all parties. Each party that doesn't receive a value within Δ time chooses a default value, e.g. 0. They then all participate in the Crusader Consensus protocol with their received value as input, and output their output from the Crusader Consensus protocol. If the sender is nonfaulty with the input x, then all nonfaulty parties will receive that value in Δ time. They then all participate the Crusader Consensus protocol with the input x, and therefore

from the Validity property output x as well. In addition, if two nonfaulty parties output $y, y' \in \{0, 1\}$, then from the Correctness property $y = y'$, as required.

4 LOWER BOUNDS

This section provides several communication lower bounds on protocols solving Binary Crusader Broadcast. The lower bounds use ideas from the Dolev-Reischuk lower bound [7], and from the subsequent work of Abraham *et al.* [1]. All of the following lower bounds are stated as lower bounds on the number of messages sent, as done in previous works. However, the lower bounds actually only use the number of messages as a bound on the number edges in the communication graph. That is, if fewer than m messages are sent in the network, then there are fewer than m pairs of parties that communicate with each other. These lower bounds cannot be avoided by increasing the number of messages without increasing the number of edges in the communication graph of the protocol. This means that the lower bounds might be more accurately stated in terms of edges in the communication graph instead of messages sent.

The first lower bound uses the fact that few messages are sent in order to isolate a single party i and cause it to communicate only with faulty parties. The faulty parties then simulate a run with the input 1 for party i when a nonfaulty sender has the input 0, causing it to output 1. The faulty parties also make sure that the rest of the network doesn't notice that i was isolated by simulating its messages in a run with the sender's input being 0 and having the faulty parties respond accordingly when communicating with other parties. Note that in order for the theorem to hold, the content of the messages actually can be probabilistic, as long as parties always communicate with the same parties throughout the protocol. All of the following bounds also include an upper bound on the number of faulty parties. This is done in order to make sure that the required number of nonfaulty parties remain in order to reach a contradiction. Clearly, if the adversary can actually corrupt a larger number of parties, it can choose not to do so and achieve the same lower bounds, slightly adjusting the exact number of messages. In all cases however, f is allowed to be a constant fraction of n.

Theorem 1. *Let there be a deterministic protocol solving Binary Crusader Broadcast in lockstep synchrony. If the protocol is resilient to f static Byzantine corruptions, then there must be at least one run of the protocol in which at least $\frac{1}{4}(n-1)f$ messages are sent for $n \geq f + 2$.*

Proof. Assume by way of contradiction that fewer than $\frac{1}{4}(n-1)f$ messages are sent overall throughout any run of the protocol. Let W_0 be a run in which the adversary does not corrupt any party and the sender has input 0. Similarly, let W_1 be a run in which the adversary does not corrupt any party and the sender has input 1. From the Validity property of the protocol all parties output 0 in W_0 and 1 in W_1. By assumption, the total number of messages sent in either run is less than $\frac{1}{4}(n-1)f$, and thus the total number of messages in both runs is less than $\frac{1}{2}(n-1)f$. Now assume by way of contradiction that at least $n-1$ parties send or receive at least f messages in total in both runs. When summing over the messages sent or received by all parties, each message is counted twice: once when it is sent and once when it is received. Therefore, the total number of messages sent in both W_0 and W_1 is at least $\frac{1}{2}(n-1)f$, reaching a contradiction to the stated above. This means that at least 2 parties send or receive no more than f messages in total in both runs. Let i be one of those parties such that i is not the sender s. Let P_0, P_1 be the sets of parties with which i communicated in W_0 and W_1 respectively. By the stated above, $|P_0 \cup P_1| \leq f$.

Now observe the run W_{hybrid} in which s has the input 0 and the adversary acts according to the following strategy: the adversary corrupts all parties in $P_0 \cup P_1$, all of those parties communicate with all parties that aren't i as nonfaulty parties would in the protocol, and communicate with i as nonfaulty parties would if s had the input 1. More precisely,

parties in $P_0 \cup P_1$ simulate all of i's messages internally when communicating with all parties other than i and act as if they received those messages, but don't send resulting messages to i. On the other hand, when communicating with i they simulate all of the messages from all other parties with a nonfaulty s having the input 1 and act accordingly, but only send resulting messages to party i. Note that both in W_0 and in W_1, all parties not in $P_0 \cup P_1 \cup \{i\}$ don't communicate directly with party i. All nonfaulty parties see communication that is identical to the one in W_0 and since they are not in P_0, they don't send any messages to i in W_{hybrid} as well. Similarly, i sees communication that is identical to the one in W_1 and thus doesn't send any messages to parties other than those in P_1 in W_{hybrid}. Therefore the view of all parties not in $P_0 \cup P_1 \cup \{i\}$ is identical to their view in W_0, and thus as stated above, they all output 0. On the other hand, i's view is identical to its view in W_1, and thus it outputs 1. Note that $n \geq f + 2$, so there are at least two nonfaulty parties. Party i and all parties not in $P_0 \cup P_1 \cup \{1\}$ are nonfaulty, so this is a violation of the Correctness property of the protocol, reaching a contradiction and completing the proof. □

The second lower bound uses ideas from [1] and generalizes them to the task of probabilistic Crusader Broadcast. The first part of the lower bound shows that if no more than $\frac{\epsilon}{4}f^2$ messages are sent in expectation in the protocol, then there is at least one non-sender party that communicates with a small number of parties with probability ϵ. Using this insight, an adversary can isolate that party and perform a similar attack to the one described in the previous theorem. The last part of the theorem shows that the probability that if the original protocol is purported to be $(\frac{2}{3} + \epsilon)$-correct, then the isolated party and the rest of the nonfaulty parties output different values with at least $\frac{1}{3}$ probability, reaching a contradiction.

Theorem 2. *Let there be a probabilistic $(\frac{2}{3} + \epsilon)$-correct protocol solving Binary Crusader Broadcast in lockstep synchrony for some $\epsilon \in \left(0, \frac{1}{3}\right]$. If the protocol is resilient to f strongly adaptive Byzantine corruptions, then the expected number of messages sent in the protocol is at least $\frac{\epsilon}{4}(n-1)f$ for $n \geq f + 2$.*

Proof. Assume that is not the case. This means that there exists a $(\frac{2}{3} + \epsilon)$-correct Binary Crusader Broadcast protocol with expected message complexity smaller than $\frac{\epsilon}{4}(n-1)f$. Similarly to the previous theorem, we will define W_0 and W_1 as runs in which the adversary does not corrupt any party and the sender has inputs 0 and 1 respectively. In both of these worlds, the probability that all parties terminate and output the sender's input must be at least $\frac{2}{3} + \epsilon$. Define M_0 and M_1 to be random variables indicating the number of messages sent by nonfaulty parties in W_0 and W_1 respectively. In addition, define $M = M_0 + M_1$ to be the number of messages sent in both runs. By assumption, $\mathbb{E}[M] = \mathbb{E}[M_0] + \mathbb{E}[M_1] < \frac{\epsilon}{2}(n-1)f$. For every $i \in [n]$, let X_i be a random variable indicating the total number of messages sent or received by party i in total both in W_0 and in W_1. Assume by way of contradiction that for at least $n-1$ parties $i \in [n], \mathbb{E}[X_i] > \epsilon f$. First note that $M = \frac{1}{2}\sum_{i=1}^{n} X_i$ because when summing over all the messages that each party sent and received, we count every message twice. Therefore, $\mathbb{E}[M] = \frac{1}{2}\sum_{i=1}^{n}\mathbb{E}[X_i] > \frac{\epsilon}{2}(n-1)f$, in contradiction. Therefore, there exist at least two parties $i, j \in [n]$ for which $\mathbb{E}[X_i], \mathbb{E}[X_j] \leq \epsilon f$. Let i be a non-sender party for which $\mathbb{E}[X_i] \leq \epsilon f$. From the Markov inequality, $\Pr[X_i \geq f] \leq \frac{\mathbb{E}[X_i]}{f} \leq \frac{\epsilon f}{f} = \epsilon$.

We will now define an adversary's attack in W_{hybrid}. The sender s has the input 0. Whenever a party j sends a message to party i, the adversary corrupts j and erases the message. In addition, whenever i sends a message to party j, the adversary corrupts j. In parallel, the adversary simulates party i's responses in W_0, given all of the messages it was sent. If party i ever sends a message to party j in that simulation in a given round, the adversary corrupts party j, erases its outgoing messages for that round, and makes it act as a nonfaulty party would if it received all of the messages it already received and the messages sent by i in the simulated run. Finally, the adversary simulates all of the

communication between all parties in W_1 given the messages sent by i in W_{hybrid}. This is done by internally running all parties in each round of the protocol except i, and using i's messages in each round. Whenever a party j sends i a message in the simulated run of W_1, the adversary corrupts it in W_{hybrid} and sends that message to i. If at any point the adversary is required to corrupt more than f parties, it aborts. Before analyzing the probability that the attack succeeds, we will define several random variables. Let A_0 be the event that all nonfaulty parties except i output 0 in W_{hybrid}. Let A_1 be the event that i outputs 1 in W_{hybrid}. Similarly, let B_0 be the event that all nonfaulty parties except i output 0 in W_0, and let B_1 be the event that i outputs 1 in W_1. Note that the definitions of A_0, B_0 allows i to output 0 as long as all other nonfaulty parties output 0. Define G to be the event that no more than f parties communicate with i in total in W_0 and W_1 combined. Finally, define G_{hybrid} to be the event that the adversary does not abort in W_{hybrid}.

Our goal is to show that $\Pr[A_0 \cap A_1] > \frac{1}{3} - \epsilon$. This contradicts the fact that the protocol is $(\frac{2}{3} + \epsilon)$-correct, because with more than $\frac{1}{3} - \epsilon$ probability, all honest parties except for i output 0, and i outputs 1. By assumption $n \geq f + 2$, so there actually are at least two nonfaulty parties. Before doing so, note that as long as the adversary isn't required to corrupt more than f parties, the view of all nonfaulty parties except i in W_{hybrid} is identical to the view they would have in W_0, given that no more than f parties communicate with i in both W_0 and W_1. Similarly, as long as that event doesn't happen, i's view is identical to the view it would have in W_1, given that no more than f parties communicate with i in both W_0 and W_1. Therefore, we know that $\Pr[G] = \Pr[G_{hybrid}]$, $\Pr[A_0|G_{hybrid}] = \Pr[B_0|G]$ and $\Pr[A_1|G_{hybrid}] = \Pr[B_1|G]$. We are now ready to analyze $\Pr[A_0 \cap A_1]$:

$$
\begin{aligned}
\Pr[A_0 \cap A_1] &= \Pr[A_0] + \Pr[A_1] - \Pr[A_0 \cup A_1] \\
&\geq \Pr[G_{hybrid}] \Big(\Pr[A_0|G_{hybrid}] + \Pr[A_1|G_{hybrid}] \Big) - 1 \\
&= \Pr[G] \Big(\Pr[B_0|G] + \Pr[B_1|G] \Big) - 1 \\
&= \Pr[B_0 \wedge G] + \Pr[B_1 \wedge G] - 1 \\
&= \Pr[B_0] - \Pr[B_0 \wedge \overline{G}] + \Pr[B_1] - \Pr[B_1 \wedge \overline{G}] - 1 \\
&\geq \Pr[B_0] + \Pr[B_1] - 2\Pr[\overline{G}] - 1 \\
&= \Pr[B_0] + \Pr[B_1] - 2\Pr[X_i > f] - 1 \\
&\geq (\frac{2}{3} + \epsilon) + (\frac{2}{3} + \epsilon) - 2\epsilon - 1 = \frac{1}{3} > \frac{1}{3} - \epsilon \,,
\end{aligned}
$$

reaching a contradiction, and completing the proof. □

The main insight of the previous theorem was that if fewer than $\Omega(nf)$ messages are sent in a protocol in expectation, then there is a good probability that at least one party communicates with f parties or fewer, and can be isolated. The next lower bound generalizes this insight and shows that if for some $c \in [0, 1]$ fewer than $\Omega(nf^{1-c})$ messages are sent in expectation, there exist f^c parties that can be isolated. From this point, the proof is extremely similar to the one of the previous theorem. Note that the exact same techniques can be used in the deterministic case with a static adversary, but the theorem is omitted due to its similarity. It is also important to note that similar theorems with different choices instead of $f^c - 1$ can easily be formulated for more general results. This specific choice was made as it simplifies some calculations, and it is enough to show that as the number of messages approaches a $O(\epsilon n)$, the number of isolated parties approaches $\Omega(f)$.

Theorem 3. *Let there be a probabilistic $(\frac{2}{3}+\epsilon)$-correct protocol solving all-but (f^c-1) Binary Crusader Broadcast in lockstep synchrony for some $c \in [0,1]$ and $\epsilon \in \left(0, \frac{1}{3}\right]$. If the protocol is resilient to $f < \frac{n}{3}$ strongly adaptive Byzantine corruptions, then the expected number of messages sent in the protocol is at least $\frac{\epsilon}{8}(n-1)f^{1-c}$ for $n \geq 3f$*[1].

Proof. Assume that is not the case. This means that there exists a $(\frac{2}{3}+\epsilon)$-correct all-but (f^c-1) Binary Crusader Broadcast protocol with expected message complexity smaller than $\frac{\epsilon}{8}(n-1)f^{1-c}$. Similarly to the previous theorem, we will define W_0 and W_1 as runs in which the adversary does not corrupt any party and the sender has inputs 0 and 1 respectively. In both of these worlds, the probability that all parties terminate and output the sender's input must be at least $\frac{2}{3}+\epsilon$. Define M_0 and M_1 to be random variables indicating the number of messages sent by nonfaulty parties in W_0 and W_1 respectively. In addition, define $M = M_0 + M_1$ to be the number of messages sent in both runs. By assumption, $\mathbb{E}[M] = \mathbb{E}[M_0] + \mathbb{E}[M_1] < \frac{\epsilon}{4}(n-1)f^{1-c}$. Similarly to before, the adversary will seek a set of $\lfloor f^c \rfloor > f^c - 1$ parties that don't contain the sender and don't send many messages. In order to do that, assume without loss of generality that the sender is party n. Let $m = \lfloor f^c \rfloor$, $\ell = \lceil \frac{n-1}{m} \rceil$, and define ℓ sets of m parties as follows: $\forall i \in \{0,\ldots,\ell-2\}$ $P_i = \{i \cdot m + 1, \ldots, (i+1)m\}$ and $P_{\ell-1} = \{n-m, \ldots, n-1\}$. We would like to guarantee that the sender is not in any of the sets P_i, and that every other party appears in one of the sets, but in no more than two of the sets. First note that the sender is not in $P_{\ell-1}$ by definition. The largest number in any of the other P_i sets is $(\ell-2+1)m$. Using the definition of ℓ, $(\ell-2+1)m = (\lceil \frac{n-1}{m} \rceil - 1)m \leq \frac{n-1}{m} \cdot m < n$, and thus the sender (party n) is not in any of those sets. Secondly, note that all of the sets up to $P_{\ell-2}$ are disjoint. This means that every party appears at most once in one of the sets $P_0, \ldots, P_{\ell-2}$ and at most once more in $P_{\ell-1}$. Finally, the sets $P_0, \ldots, P_{\ell-2}$ exactly contain the parties $1, \ldots, (\ell-2+1)m$. Note that $(\ell-2+1)m = (\lceil \frac{n-1}{m} \rceil - 1)m \geq (\frac{n-1}{m} - 1)m = n-1-m$, and thus $P_{\ell-1}$ contains all of the rest of the parties, except for the sender.

As defined in the previous lower bound, for every $i \in [n]$, let X_i be a random variable indicating the total number of messages sent or received by party i in total both in W_0 and in W_1. In addition, for every $i \in \{0,\ldots,\ell-1\}$ let Y_i be the total number of messages sent or received by all parties $j \in P_i$ in total both in W_0 and in W_1. It is always the case that $\sum_{j \in P_i} X_j \geq Y_i$ because $\sum_{j \in P_i} X_j$ counts all messages sent or received by parties in P_i, and might even count some of those messages twice. Assume by way of contradiction that for every $i \in \{0,\ldots,\ell-1\}$, $\mathbb{E}[Y_i] > \epsilon f$. First note that $M = \frac{1}{2}\sum_{i=1}^{n} X_i$ because when summing over all the messages that each party sent and received, we count every message twice. In addition, seeing as each party j appears in at most two of the sets P_i, $2\sum_{i=1}^{n} X_i \geq \sum_{i=0}^{\ell-1} \sum_{j \in P_i} X_j$.

[1] It is actually enough that $n \geq f + 2f^c$, since all we need is f faulty parties and 2 sets of at least f^c nonfaulty parties to disagree on the output.

Combining these observations:

$$\begin{aligned}
\mathbb{E}[M] &= \mathbb{E}[\frac{1}{2}\sum_{i=1}^{n} X_i] \\
&= \frac{1}{4}\mathbb{E}[2\sum_{i=1}^{n} X_i] \\
&\geq \frac{1}{4}\mathbb{E}[\sum_{i=0}^{\ell-1}\sum_{j\in P_i} X_j] \\
&\geq \frac{1}{4}\sum_{i=0}^{\ell-1}\mathbb{E}[Y_i] \\
&\geq \frac{1}{4}\ell\epsilon f \\
&= \frac{1}{4}\lceil\frac{n-1}{m}\rceil\epsilon f \\
&\geq \frac{1}{4}\cdot\frac{n-1}{\lfloor f^c\rfloor}\epsilon f \\
&\geq \frac{1}{4}\frac{n-1}{f^c}\epsilon f = \frac{\epsilon}{4}(n-1)f^{1-c}
\end{aligned}$$

in contradiction. This means that there exists at least one $k \in \{0,\ldots,\ell-1\}$ for which $\mathbb{E}[Y_k] \leq \epsilon f$. Let P_k be such a set. From the Markov inequality, $\Pr[Y_k \geq f] \leq \frac{\mathbb{E}[Y_k]}{f} \leq \frac{\epsilon f}{f} = \epsilon$. In other words, the probability that in total all parties in P_k send and receive more than f messages in W_0 and in W_1 combined is no greater than ϵ.

We will now define an adversary's attack in W_{hybrid}, similar to the attack in Theorem 2. The sender s has the input 0. Whenever a party $j \notin P_k$ sends a message to a party $i \in P_k$, the adversary corrupts j and erases the message. In addition, whenever a party $i \in P_k$ sends a message to a party $j \notin P_k$, the adversary corrupts j. In parallel, the adversary simulates all of the messages parties $i \in P_k$ send in W_0, given all of the messages they were sent by parties not in P_k. If any party $i \in P_k$ ever sends a message to party $j \notin P_k$ in that simulation in a given round, the adversary corrupts party j, erases its outgoing messages for that round, and makes it act as a nonfaulty party would if it received all of the messages it already received and the messages sent by all parties in P_k in the simulated run. Finally, the adversary simulates all of the communication between all parties in W_1 given the messages sent by all parties $i \in P_k$ in W_{hybrid}. This is done by internally running all parties in each round of the protocol except for parties in P_k, and using the messages sent by parties in P_k in each round. Whenever a party $j \notin P_k$ sends some party $i \in P_k$ a message in the simulated run of W_1, the adversary corrupts j in W_{hybrid} and sends that message to i. If at any point the adversary is required to corrupt more than f parties, it aborts. The adversary never corrupts any party $i \in P_k$, so all parties in P_k remain nonfaulty. Before analyzing the probability that the attack succeeds, we will define several random variables. Let A_0 be the event that all nonfaulty parties except parties in P_k output 0 in W_{hybrid}. Let A_1 be the event that all parties in P_k output 1 in W_{hybrid}. Similarly, let B_0 be the event that all nonfaulty parties except parties in P_k output 0 in W_0, and let B_1 be the event that all parties in P_k output 1 in W_1. Note that the definitions of A_0, B_0 allow all parties in P_k to output 0, as long as all other nonfaulty parties do so as well. Define G to be the event that no more than f parties communicate with parties in P_k in total in W_0 and W_1 combined. Finally, define G_{hybrid} to be the event that the adversary does not abort in W_{hybrid}.

Our goal is to show that $\Pr[A_0 \cap A_1] > \frac{1}{3} - \epsilon$. Note that in this case, all parties in P_k output 1 in W_{hybrid} and all other nonfaulty parties output 0. There are f^c parties in P_k and at least $n - f - f^c \geq n - 2f \geq f \geq f^c$ nonfaulty

parties not in P_k. Therefore, with probability greater than $(\frac{1}{3} - \epsilon)$ at least f^c nonfaulty parties output 0 and at least f^c nonfaulty parties output 1, contradicting the fact that the protocol is an $(\frac{2}{3} + \epsilon)$-correct all-but $(f^c - 1)$ Binary Crusader Broadcast protocol. Before doing so, note that as long as the adversary isn't required to corrupt more than f parties, the view of all nonfaulty parties except parties in P_k in W_{hybrid} is identical to the view they would have in W_0, given that no more than f parties communicate with parties in P_k in both W_0 and W_1. Similarly, as long as that event doesn't happen, the view of all parties in P_k in W_{hybrid} is identical to the view they would have in W_1, given that no more than f parties communicate with parties in P_k in both W_0 and W_1. Therefore, we know that $\Pr[G] = \Pr[G_{hybrid}]$, $\Pr[A_0|G_{hybrid}] = \Pr[B_0|G]$ and $\Pr[A_1|G_{hybrid}] = \Pr[B_1|G]$. We are now ready to analyze $\Pr[A_0 \cap A_1]$:

$$\begin{aligned}
\Pr[A_0 \cap A_1] &= \Pr[A_0] + \Pr[A_1] - \Pr[A_0 \cup A_1] \\
&\geq \Pr[G_{hybrid}] \Big(\Pr[A_0|G_{hybrid}] + \Pr[A_1|G_{hybrid}]\Big) - 1 \\
&= \Pr[G]\Big(\Pr[B_0|G] + \Pr[B_1|G]\Big) - 1 \\
&= \Pr[B_0 \wedge G] + \Pr[B_1 \wedge G] - 1 \\
&= \Pr[B_0] - \Pr[B_0 \wedge \overline{G}] + \Pr[B_1] - \Pr[B_1 \wedge \overline{G}] - 1 \\
&\geq \Pr[B_0] + \Pr[B_1] - 2\Pr[\overline{G}] - 1 \\
&= \Pr[B_0] + \Pr[B_1] - 2\Pr[Y_k > f] - 1 \\
&\geq (\frac{2}{3} + \epsilon) + (\frac{2}{3} + \epsilon) - 2\epsilon - 1 = \frac{1}{3} > \frac{1}{3} - \epsilon \,,
\end{aligned}$$

reaching a contradiction, and completing the proof. □

5 ECLIPSE ATTACKS IN BLOCKCHAIN SYSTEMS

Recent years have seen wide interest in blockchain systems, both in practice and in theory. As such, the security of such systems has been widely researched as well. Much of the research in this field focused on the security of the protocols assuming that once sent, blocks propagate quickly throughout the network, for example in [4, 9, 14, 16]. Other parts of the literature focus on researching the underlying peer-to-peer communication network in many real-world blockchain systems and the ways it might affect the security of the system as a whole. Some of these works dealt with "eclipse attacks" [11, 15], in which an adversary isolates a specific party (or group of parties), and filters its communication in advantageous ways.

While important and foundational to the body of blockcahin research, these works have often ignored the connections between current security threats and classic results known in the world of distributed systems. One such important result is that of Dolev and Reischuk [7] (and its extensions by Abraham *et al.* [1]). These works provide communication lower bounds for implementing broadcast algorithms, which are known to reduce to consensus algorithms as well [5]. The lower bounds show that at least a quadratic number of messages needs to be sent in order to achieve a broadcast protocol. The proofs uses the fact that in the classic definition of broadcast protocols, parties are always required to eventually output some value. This is true even if they don't hear anything throughout the whole run (silent run). This means that it is enough to isolate a party and drop all communication to it in order to break safety. The isolated party has to choose some value to output in a manner independent from what the rest of the network sees. Choosing the party to isolate and setting up the rest of the network in a specific manner then causes different nonfaulty parties to output different values.

The lower bounds of Dolev and Reischuk, and its extensions and all of the lower bounds shown in Section 4 actually construct an adversary that attacks the protocol, assuming not enough messages are sent. We suggest viewing the known eclipse attacks as specific instances of the attacks described in the Dolev-Reischuk paper and in related lower bounds. This allows to more systematically consider this type of attack and possibly try to find general solutions as opposed to patching the specific attack found in a specific paper. As suggested in [11, 15], after isolating a node (or several nodes), the adversary can perform several types of attacks. Some of these attacks are direct attacks on the consensus protocol, e.g. a double-spend attack. More precisely, nodes briefly locally confirm different conflicting transactions, each thinking that the transaction they output took place. This is entirely analogous to outputting different values in the attacks shown in this work, in which parties output different non-$\perp$ values from the crusader broadcast protocol. Even more extreme, [11, 15] suggest using the eclipse attack to then perform a 51% attack, allowing an adversary to repeatedly perform such double-spend attacks, as well as other attacks. It is important to note that eclipsing communication can also allow an adversary to perform additional attacks that aren't directly attacks on the agreement between parties. For example, as suggested in [11], an attacker could eclipse a portion of the network in order to perform selfish mining attacks [4, 9]. In the suggested attack, the adversary would filter communication to and from the eclipsed nodes, and use the nodes' mining power to its advantage. This attack is also strikingly similar to the one described in Theorem 3, which suggests that these lower bounds could be of interest also when not directly attacking the agreement of the protocol, but rather notions like liveness or fairness of a consensus protocol.

Unfortunately, the classic Dolev-Reischuk attack has a slightly annoying feature: it relies on silence. It assumes that parties that don't hear anything must output some value and cannot remain undecided. This is of course not the case in some real world systems. For example, in cryptocurrency systems, a node wouldn't suddenly decide that a transaction took place without seeing a block including that transaction. For that reason, in this paper we chose to focus on the task of Crusader Broadcast instead. In Crusader Broadcast, parties are allowed to output a special value $\perp$, that is not a possible input to the system. This can be thought of as choosing to remain undecided, or not to agree on a specific value. One important caveat is that a natural desired property in such a protocol is that a value broadcasted by a nonfaulty sender should be received and agreed upon by all parties eventually. The lower bounds of Theorems 1 and 2 show that the classic result of Dolev and Reischuk can be generalized to this task as well both for deterministic and probabilistic protocols. It is important to note that a big part of the difference between the attacks described in Theorems 1 and 2 stems from the probabilistic nature of the communication graph, and not from difference in the content of messages. The proof of Theorem 1 would not change in a probabilistic protocol for a 1-correct Binary Crusader Broadcast protocol with a static communication graph. In addition, Theorem 3 generalizes the result even further and shows that as the number of messages decreases, or more precisely the number of edges in the communication graph decreases, a larger number of nonfaulty parties can be isolated and made to output a different value. Note that as the number of edges in the communication graph tends towards $O(\epsilon \cdot n)$, the number of isolated parties tends towards $\Omega(f)$. This allows a large adversary to partition the nonfaulty parties into two large groups that disagree on the output of the protocol.

5.1 Limitations of Real-World Adversaries

The attacks descibed in Section 4 assume extremely strong adversaries. First of all, in all lower bounds, the adversary is assumed to be able to simulate other parties. This assumption does not hold in real-world systems. For example, adversaries have limited compute-power. This means that they generally cannot arbitrarily simulate other parties in proof-of-work systems. Furthermore, in systems with a public key infrastructure, adversaries cannot forge other parties' signatures or break other cryptographic primitives during the simulation of the protocol. The adversary in Theorems 2

and 3 is also assumed to be strongly adaptive. In the real world, adversaries generally cannot corrupt parties at will, let alone retroactively delete their messages and replace them. Given all of these limitations, one could reasonably ask: *are the attacks described in these lower bounds even applicable to the real world?*

Surprisingly, the answer seems to be that they are applicable to the real world, as evidenced by previous works on eclipse attacks. At first we will tackle the need for strong adaptivity. As shown in [11, 15], in both Bitcoin's and Ethereum's peer-to-peer communication protocols an adversary can monopolize a node's connections. When nodes restart, they initiate outgoing connections from tables storing addresses of known peers. In order to monopolize a node's connections, an adversary fills its tables in advance with the addresses of nodes controlled by the adversary, and then causes it to restart. After restarting, the node will choose nodes from those tables, and potentially only connect to the adversary's nodes. In addition, nodes may receive incoming connections from peers. After causing a node to restart, the adversary also sends incoming connection requests and monopolizes all of the incoming connections. These attacks are performed in advance, allowing the adversary to essentially structure the communication graph in an advantageous manner. When relating these attacks to the setting of the lower bounds above, the adversary does not need to be adaptive, let alone strongly adaptive. Even worse, the lower bound in Theorem 1 only shows that there is some party that can be isolated. In that attack the adversary has to have some special knowledge of that specific party and tailor its attack to it. On the other hand, in the real-world attacks described in [11, 15], the adversary can choose whichever node it wants and isolate it in a static manner, without the need to find out which node can be isolated.

The second problematic assumption on the adversary's power is the ability to simulate. In the lower bounds above, that requirement stems from the fact that we do not know what the parties may do in the protocol. For example, parties may use cryptography in order to guarantee that a large portion of the network saw some value (see [2, 20] for such examples). In order to fully simulate the behaviour of the nonfaulty parties, an adversary would have to be able to break some of the cryptographic assumptions made in the design of the protocol. On the other hand, in many current blockchain systems simulating the actions required in the consensus "only" entails mining blocks with the correct information. Of course an adversary is limited by its own compute-power, so it can't actually fully simulate the rest of the network for the isolated parties. However, some of the uses for eclipse attacks have suggested ways to mitigate this issue. Note that the following attacks are well-known, and we mention them as examples of ways the adversary can simulate parts of the network as required by our lower bounds in practice. For example, eclipsing a fraction of the network could allows the adversary to effectively increase the fraction of compute power it has in the rest of the network. In this sense, the only thing the adversary needs to be able to "simulate" is some nodes going offline. As suggested in [11], this can be done gradually in order not to be detected. Even worse, the adversary could conceivably utilize honest nodes to simulate the protocol for it. This can be done by simply letting only parts of the network see a given block. The adversary could then use the fact that nodes would continue to mine on top of it as a means of simulating the work required, and then showing the mined blocks to the rest of the network when needed.

In short, at first glance the adversary described in the lower bounds of Section 4 seems too powerful to be of interest when discussing real-world systems. However, some of the biggest real-world systems used today actually don't require it to be so powerful in order to levy attacks. This suggests possible ways to defend against such an adversary. The first obvious way is simply raising the number of links in the system, as suggested in other works. In addition to this making intuitive sense, from a theoretic standpoint the conditions of the lower bounds shown above would stop holding. As suggested by [11, 15], measures could be taken in order to make it harder to fill the outgoing ink tables with the adversary's nodes' addresses. In that case, Theorem 1 suggests that if the number of edges in the communication graph isn't large enough, a large enough adversary should be able to isolate some party. One could think of ways to

require the adversary to actually be strongly adaptive in order to eclipse communications. This could take form in more extreme countermeasures, such as having a dynamically changing communication graph with outgoing edges being chosen randomly. In that case, an adversary seeking to isolate a node would have to adaptively corrupt all random incoming and outgoing links to it, which would be prohibitively expensive.

Alternatively, one could make it harder to simulate parts of the protocol. For example, this could be done by requiring more nodes to sign blocks, or by making more use of cryptography in the communication layer itself. Note that simply requiring a number of signatures would not be enough because the adversary could generate many signing keys and provide the required signatures. As is standard in preventing such Sybil attacks, the signatures should represent a portion of the compute-power in the network instead. Such defenses should be very carefully implemented seeing as nodes could then refuse to sign certain blocks, or simply go offline, impeding the progress of the honest network. Previous works, such as ByzCoin [13] have suggested ways to implement protocols that rely on a large enough number of parties signing blocks. However, these protocols are still susceptible to network partitioning attacks.

6 SUBQUADRATIC CRUSADER BROADCAST PROTOCOL

In this section, we start by presenting a quadratic crusader broadcast protocol for ease of explanation. A second protocol is then constructed, using ideas from the first one. In the second protocol, an expensive all-to-all round is replaced by a gossip procedure [17], lowering the overall communication cost in a standard way. Note that crucially the gossip protocol is *actually* random, in the sense that parties choose peers randomly in *every* round. This gossip protocol is secure and efficient against a static adversary. However, in order to be secure against an adaptive adversary, it has to send a larger number of messages. In addition, we assume the existence of a public key infrastructure (PKI), used in a signature scheme. An adversary being able to simulate messages from the sender needs to be able to break the signature scheme, leading to it breaking the validity of the protocol. In other words, in some sense this protocol is a sort of "minimal example" showing that it is easy to force the adversary to either be adaptive or to be able to simulate other parties in order to break subquadratic crusader broadcast protocols.

The simplified $O(n^2)$ protocol, presented in Algorithm 1, proceeds in two rounds. In the first round, the sender s sends a signed message with its input to all parties. Parties then inform each other of the message they've seen. Finally, any party that received a message m from the sender without seeing any conflicting message outputs m. If either of these conditions doesn't hold, that party outputs $\perp$ instead. This protocol is captured in Algorithm 1. In general for a protocol X, denote X_i to be the code for party i executing protocol X. We assume the existence of a PKI such that every party i knows a signing key sk_i and all parties know the associated public key pk_i. The PKI is used in a signature scheme consisting of the signing algorithm Sign and verification algorithm Verify. We analyze the signature scheme as perfectly secure, meaning that only i can produce signatures which verify with respect to pk_i. A similar analysis can be done allowing for a negligible probability of error (meaning that the resulting protocol is $1 - \mathsf{negl}(\lambda)$ correct, with λ being the security parameter).

The protocol consists of a single multicast requiring $O(n)$ messages, and a single all-to-all round requiring $O(n^2)$ messages. A proof of the protocol follows:

Theorem 4. *The* CrusaderBroadcast *protocol is a Crusader Broadcast protocol resilient to any number of Byzantine corruptions f in a synchronous system.*

Proof. Each property is proven individually. Denote val_i to be the variable val stored by party i.

Algorithm 1 CrusaderBroadcast$_i$

1: $val \leftarrow \perp$
2: **if** i is the sender s with input x **then**
3: $\quad \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, x)$
4: $\quad$ send the message $\langle$"sender", $x, \sigma\rangle$ to all parties
5: **wait** Δ time
6: **if** a $\langle$"sender", $m, \sigma\rangle$ message was received from s while waiting such that $\mathsf{Verify}(\mathsf{pk}_s, m, \sigma) = 1$ **then**
7: $\quad val \leftarrow m$
8: $\quad$ send $\langle$"forward", $m, \sigma\rangle$ to all parties
9: **wait** Δ time
10: **if** a $\langle$"forward", $m', \sigma'\rangle$ message was received while waiting such that $m' \neq val$ and $\mathsf{Verify}(\mathsf{pk}_s, m', \sigma') = 1$ **then**
11: $\quad val \leftarrow \perp$
12: **output** val and **terminate**

Validity. Assume the sender s is nonfaulty with input x. In the beginning of the protocol it produces a signature σ for x, and sends the message $\langle$"sender", $x, \sigma\rangle$ to all parties. Every nonfaulty party receives that message up to Δ time after that, and updates val to x. The sender didn't sign any other value $m' \neq x$, so no nonfaulty party will receive a $\langle$"forward", $m', \sigma'\rangle$ message with such that $m' \neq val$ and $\mathsf{Verify}(\mathsf{pk}_i, m', \sigma') = 1$. Therefore no nonfaulty party reverts val back to $\perp$. Finally, after 2Δ time, all nonfaulty parties output $val = x$ and terminate.

Correctness. Assume by way of contradiction two nonfaulty parties $i \neq j$ output two non-$\perp$ values m_i, m_j respectively such that $m_i \neq m_j$. Those parties output the variable val at the end of the protocol, after 2Δ time. By assumption, they output non-$\perp$ values, so $val_i \neq \perp$ and $val_j \neq \perp$. Party i only updates val_i to $m_i \neq \perp$ at time Δ in line 7, if it received a $\langle$"sender", $m_i, \sigma_i\rangle$ message from s such that $\mathsf{Verify}(\mathsf{pk}_s, m_i, \sigma_i) = 1$. It then sends the message $\langle$"forward", $m_i, \sigma_i\rangle$ to all parties at time Δ. Party j receives that message by time 2Δ, sees that $m_i \neq m_j$ and $\mathsf{Verify}(\mathsf{pk}_s, m_i, \sigma_i) = 1$ and updates val_j to $\perp$. Finally, j outputs $val_j = \perp$, contradicting the fact that it output some value $m_j \neq \perp$.

Termination. All parties wait for 2Δ overall and terminate. □

Building on the CrusaderBroadcast protocol, we can replace the expensive all-to-all round with a more efficient randomized procedure. The protocol is parameterized by two constants c, d. In the protocol, parties gossip for $O(c \log n)$ rounds and communicate with $O(d \log n)$ parties in expectation in each round. These values can be adjusted to increase the probability of success of the protocol. This results in the protocol described in Algorithm 2.

Note that the proof of Theorem 4 could be used in its entirety to prove the security of Algorithm 2 with one exception: the proof of the Correctness property relies on the fact that if a nonfaulty party i updates its val variable to m, then every nonfaulty party will see that value before terminating. This means that in order to prove the security of Algorithm 2, we need to show that if a nonfaulty party updates val to m, then every nonfaulty party will add m to $received$ before terminating with all but a small probability p. Using the union bound, this will mean that with probability $1 - np$, every nonfaulty party will add val_i to the set $received$ for every nonfaulty i before terminating. From this point on, the rest of the proof is identical, yielding a $(1 - np)$-correct Crusader Broadcast protocol.

In the following discussion, we will use well-known results [6, 12, 17] regarding gossip protocols. In short, assume some party has a value v to spread in a network of nonfaulty parties. We know that if every party that knows the value v sends it to one random party every round for $c \log n$ rounds, then all parties know the value v by round $c \log n$ with

Algorithm 2 CrusaderGossip$_i(c, d)$

1: $val \leftarrow \perp, received \leftarrow \emptyset$
2: **if** i is the sender s with input x **then**
3: $\quad \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, x)$
4: $\quad$ send the message $\langle$"sender", $x, \sigma\rangle$ to all parties
5: **wait** Δ time
6: **if** a $\langle$"sender", $m, \sigma\rangle$ message was received from s while waiting such that $\mathsf{Verify}(\mathsf{pk}_s, m, \sigma) = 1$ **then**
7: $\quad val \leftarrow m, received \leftarrow received \cup \{(m, \sigma)\}$
8: **upon** receiving a $\langle$"forward", $forwarded\rangle$ message, **do**
9: $\quad$ **if** $\forall (m', \sigma') \in forwarded$ $\mathsf{Verify}(\mathsf{pk}_s, m', \sigma') = 1$ **then**
10: $\quad\quad received \leftarrow received \cup forwarded$
11: **for** $r \leftarrow 1, \ldots, c \log n$ **do**
12: $\quad$ **for** $j \in [n]$ **do**
13: $\quad\quad$ send $\langle$"forward", $received\rangle$ to j with probability $\frac{2d \ln n}{n}$
14: $\quad$ **wait** Δ time
15: **if** $\exists (m', \sigma') \in received$ $s.t.$ $m' \neq val$ **then**
16: $\quad val \leftarrow \perp$
17: **output** val and **terminate**

probability $1 - \frac{1}{n^{\Omega(c)}}$ or greater. Using this fact, we will then only observe the communication between nonfaulty parties and prove the following lemma:

Lemma 1. *Let the number of faulty parties be $f < \frac{n}{2}$. Assume some nonfaulty party i updates val to $m \neq \perp$ and adds (m, σ) to its $received_i$ set in line 7. The probability that there exists some nonfaulty party j that doesn't have (m, σ) in its $received_j$ set by the time it terminates is $O(\frac{1}{p(n)})$ for some polynomial $p(n)$.*

Proof. First, we will bound the probability that there exists some nonfaulty party that doesn't send at least one message to a nonfaulty party in every round. Let $h = n - f$ be the number of nonfaulty parties. By assumption $f < \frac{n}{2}$, so $h > \frac{n}{2}$. Observing a single round, the probability that a single nonfaulty party doesn't send a message to at least one nonfaulty party in a single round can be bounded by:

$$\left(1 - \frac{2d \ln n}{n}\right)^h \leq \left(1 - \frac{2d \ln n}{n}\right)^{\frac{n}{2}} \leq e^{-d \ln n} = n^{-d}$$

Using the union bound, the probability that there exists a nonfaulty party that doesn't send a message to another nonfaulty in a given round is no greater than n^{-d+1}. Using the union bound again, the probability that there exists a round in which there exists a nonfaulty party that doesn't send at least one message to at least one nonfaulty party is no greater than $\frac{c \log n}{n^{d-1}}$. Note that a nonfaulty party sends messages to each party with the same probability, meaning that the message is sent to at least one uniformly sampled nonfaulty party.

Nonfaulty parties only add pairs (m', σ') to their $received$ sets after checking that $\mathsf{Verify}(\mathsf{pk}_s, m', \sigma') = 1$. Therefore, if a nonfaulty party receives a $\langle$"forward", $forwarded\rangle$ message from a nonfaulty party, it will add all values from $forwarded$ to its $received$ set. Now, observe only the communication between nonfaulty parties that contains the pair (m, σ) described in the lemma. Assume that in every round every nonfaulty party sends at least one message to some uniformly sampled nonfaulty party. Note that sending more than one such message cannot hurt the spread of the

information. In this case, as described above, every nonfaulty party will be informed of the pair (m, σ) by round $c \log n$ and add it to its *received* set with probability $1 - \frac{1}{n^{\Omega(c)}}$ or greater. Combining the two failure probabilities, we find that the probability that some nonfaulty party doesn't have the pair (m, σ) in its received set by the time it terminates is no greater than $\frac{c \log n}{n^{d-1}} + \frac{1}{n^{\Omega(c)}}$, as required. □

Using the previous lemma and the union bound, the probability that there exists a nonfaulty party that updates *val* to $m \neq \perp$ such that not all nonfaulty parties are informed of m by the end of the protocol is no greater than $\frac{c \log n}{n^{d-2}} + \frac{1}{n^{\Omega(c)-1}}$. Setting c such that the $\Omega(c)$ term is at least 2, and d to be 4, we get that the probability of failure is $O(\frac{1}{n})$. In other words, for every $\epsilon > 0$, there exists a large enough n such that the CrusaderGossip protocol is a $(1-\epsilon)$-correct Crusader Broadcast protocol, resilient to f static Byzantine corruptions if the adversary cannot simulate the nonfaulty parties. In order to get a failure probability of $O(\frac{1}{n^k})$, one could choose $c = \Theta(k)$ and $d = k + 3$ instead.

The protocol consists of one multicast round, requiring $O(n)$ messages to be sent, and $O(c \log n)$ gossip rounds. In each such round, every nonfaulty party i sends a message to party j with probability $\frac{2d \ln n}{n}$, yielding an expected $2d \ln n$ messages per round per nonfaulty party. Overall, this results in $O(cd \cdot n \log^2 n)$ messages being sent by nonfaulty parties in expectation. Choosing the c, d as the constants described above yields the message complexity of $O(n \log^2 n)$ (and for $n = 2f + 1$, a complexity of $o(nf)$).

ACKNOWLEDGMENTS

This work was supported by the HUJI Federnann Cyber Security Research Center in conjunction with the Israel National Cyber Directorate (INCD) in the Prime Minister's Office under Grant No.: 3011004045.

REFERENCES

[1] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Validated asynchronous byzantine agreement with optimal resilience and asymptotically optimal time and word communication, 2018.

[3] Ittai Abraham and Kartik Nayak. The dolev and reischuk lower bound: Does agreement need quadratic messages? https://decentralizedthoughts.github.io/2019-08-16-byzantine-agreement-needs-quadratic-messages/, 2019.

[4] Lear Bahack. Theoretical bitcoin attacks with less than half of the computational power (draft), 2013.

[5] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[6] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[7] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[8] C Dwork, D Peleg, N Pippenger, and E Upfal. Fault tolerance in networks of bounded degree. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, page 370–379, New York, NY, USA, 1986. Association for Computing Machinery.

[9] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable, 2013.

[10] Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[11] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[12] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[13] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing, 2016.

[14] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*. Washington, DC, 2013.

[15] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. *Cryptology ePrint Archive*, 2018.

[16] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[17] Boris Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

[18] Peter Robinson, Christian Scheideler, and Alexander Setzer. Breaking the $\tilde{\omega}(\sqrt{n})$ barrier: Fast consensus under a late adversary. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, page 173–182, New York, NY, USA, 2018. Association for Computing Machinery.

[19] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[20] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain, 2018.