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 (even against omission faults) requires at least a quadratic number of messages to be sent by nonfaulty parties. In contrast, many blockchain systems achieve consensus with seemingly linear communication per instance against Byzantine faults. We explore this dissonance in three main ways. First, we extend the Dolev-Reischuk family of lower bounds and prove a new lower bound for Crusader Broadcast protocols. Our lower bound for crusader broadcast requires non-trivial extensions and a much stronger Byzantine adversary with the ability to simulate honest parties. Secondly, we extend our lower bounds to all-but-m Crusader Broadcast, in which up to m parties are allowed to output a different value. Finally, we discuss the ways in which these lower bounds relate to the security of blockchain systems. We show how *Eclipse-style attacks* in such systems can be viewed as specific instances of the attacks used in our lower bound for Crusader Broadcast. This connection suggests a more systematic way of analyzing and reasoning about Eclipse-style attacks through the lens of the Dolev-Reischuk family of attacks.

CCS Concepts: • **Theory of computation** → Distributed algorithms.

Additional Key Words and Phrases: consensus, crusader broadcast, Byzantine fault tolerance, blockchain, synchrony, 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 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 is the work of Dolev and Reischuk in 1985 [9]. They prove that any deterministic protocol solving fault tolerant broadcast must send at least $\Omega(n \cdot f)$ messages, where n is the number of parties overall and f is the number of omission-faulty parties, whose incoming and outgoing messages can be dropped [1]. Since broadcast and consensus reduce to each other [6], the lower bound also provides a lower bound on consensus. Hadzilacos and Halpern [14] show that a similar lower bound also holds only when considering fault-free runs of broadcast protocols if they are designed to be resilient to faults. Abraham, Chun, Dolev, Nayak, Pass, Ren, and Shi [1] 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 $\Omega(\epsilon n f)$ messages to be sent in expectation (assuming a strongly adaptive adversary).

1.1 Dolev-Reischuk does not hold for Crusader Broadcast

A slightly relaxed task related to that of broadcast is the task of *Crusader Broadcast* [8], in which parties are allowed to remain undecided when the sender is faulty. This is formalized by allowing parties to output a special non-value, $\perp$. Two important restrictions, in this case, are that no two nonfaulty parties may output different non-$\perp$ values and that

[1] the lower bound in [9] mentioned malicious adversaries, the extension to omission failures appears in [3].

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

all nonfaulty parties must output the sender's input if it is nonfaulty. The known Dolev-Reischuk style attacks heavily rely on the fact that parties have to output some value from the protocol, regardless of what they see. This is utilized by completely isolating a party, forcing it to communicate only with omission faulty parties. The adversary then simply blocks all communication with the isolated party, forcing it to output some value without hearing anything throughout the protocol. All that is left to do is make sure that other parties output the other value, successfully attacking any protocol with low communication complexity. However, in Crusader Broadcast, a nonfaulty party is allowed to output $\perp$ if it hears nothing throughout the protocol. Since a nonfaulty sender may send messages to any party without reaching quadratic communication complexity, it is entirely possible that in any run in which the sender is nonfaulty, no party can be completely isolated from nonfaulty parties in the protocol. This implies that the Dolev-Reischuk lower bound attack does not hold as is for Crusader broadcast protocols.

1.2 A New Lower Bound for Crusader Broadcast

The main contribution of this paper is a new lower bound for crusader broadcast. It differs substantially from the classic Dolev-Reischuk lower bound in that the adversary is required to **actively** corrupt parties. Using Byzantine corruption also raises a new challenge that is similar to that of the lower bound proven by Fischer, Lynch and Meritt [12]: the corrupted parties need to be able to **simulate** honest parties.

Intuitively, while classic Dolev-Reischuk isolates one party and makes it hear no message at all, while other parties hear a sender sending say v, in our lower bound we isolate one party and make it think it is living in an alternative world where the sender is sending $v' \neq v$. Again intuitively, building an alternative universe is harder since it requires active simulation of other parties, and thus requires more malice than just causing the isolated party to hear nothing.

This type of attack (isolating a node and making it think it's living in an alternative world) is not just theoretical, we discuss is section 5 how Eclipse-type attacks can be viewed as types of this attack. We find this connection between a theoretical lower bound and Eclipse-style blockchain attacks to be a conceptual contribution of its own and expand on this in section 1.4.

1.3 Extending to the all-but-m model

Yet another problem that the classic Dolev-Reischuk lower bound does not cover is **almost everywhere agreement** [10, 17]. This notion is closely related to that of classic agreement protocols but allows a small number of nonfaulty parties to output the wrong value, as long as the percentage of those dissenting parties tends towards 0 as n tends towards infinity. In particular, classic Dolev-Reischuk just shows a safety violation of one party. What if we allow some fraction of the parties to dissent and output differing values? Does some variant of Dolev-Reischuk hold in this case?

In this work, we prove that if the number of messages sent is significantly smaller than quadratic, then a large number of parties can be made to disagree. Concretely, if the number of messages sent is $O(nf^{1-c})$ for some $c \in [0, 1]$, the number of parties that can be made to output a differing value is $O(f^c)$. Crucially, as the number of messages in the protocol approaches $O(n)$, the adversary can make $O(f)$ nonfaulty parties disagree, which is often taken to be a constant fraction of the total number of parties.

1.4 Why does Crusader Broadcast matter for Blockchains? Connections to Eclipse Style Attacks

Blockchain systems are designed to solve the task of consensus [21, 24], or more precisely, state machine replication. Many of these systems claim to achieve consensus in a **linear** number of messages per block. This seems to be in direct conflict with the lower bounds of Dolev-Reischuk, suggesting that at least **quadratic** ($\Omega(nf)$) messages are required.

One way of making sense of this contradiction is by looking at the details of the lower bounds. As discussed above, Dolev-Reischuk prove such lower bounds for protocols in which parties are required to output some value eventually, even without hearing any message. In Bitcoin and Ethereum (Nakamoto consensus based systems) not hearing new blocks simply causes parties to not decide anything. Formally, we can think of this as if such a party outputs a special value $\perp$, signifying not knowing. In this sense, it is natural to actually view such blockchain protocols, when focusing on a single block, **as solving Crusader Broadcast rather than consensus**.

In section 5, we explore the connections between our new lower bounds for crusader broadcast and the eclipse style attacks in proof-of-work blockchain systems. We observe the following similarity:

- In our lower bound for a single shot of crusader broadcast, in the deterministic case the adversary can be static to isolate a party and needs to be able to simulate other parties to trick the isolated party to believe an alternative world.
- Similarly, in Eclipse-style attacks, the systems had insufficient randomness at the network layer, meaning that the communication graph induced by the protocol was either deterministic, or very easy to influence. This allowed a relatively static adversary to isolate a party, or even worse. Similarly, by the nature of Nakamoto consensus, the adversary can simulate other parties (often for a limited time) and trick the isolated party to believe an alternative world.

Note that in both cases, the isolated party believes in an alternative world and a double spend is executed. This is unlike the classic Dolev-Reischuk lower bound where the isolated node just sees silence (so there is just one spend and one party that does not observe the spend). Finally, we discuss in section 6 the consequence of our lower bound for the study of upper bounds for crusader broadcast. In particular, we show how a subquadratic protocol for crusader broadcast takes advantage of randomization and cryptography in order to circumvent the $\Omega(n^2)$ lower bound.

Our contributions

To summarize, our work makes three main contributions:

(1) A new Lower Bound for crusader broadcast. While it is definitely part of the enhanced Dolev-Reischuk family, it requires new non-trivial extensions. In particular, Byzantine adversaries and the ability to simulate.

Theorem 4.1. *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 $n \geq f+2$.*

(2) We extend the Dolev-Reischuk style lower bounds to the **all but *m* model**, showing that near linear protocols may actually suffer a near linear number of isolated parties. 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 4.2.

Theorem 4.3. *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 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 $n \geq 3f$.*

(3) Make the conceptual connection between Eclipse style attacks and our new Crusader broadcast lower bound. We believe that by highlighting this connection, protocol designers may be able to more rigorously design blockchain protocols that are more secure against Eclipse-style attacks.

2 COMMUNICATION AND ADVERSARY MODEL

We consider a fully-connected network of n parties with synchronous communication: there is a commonly known bound Δ on message delay. The adversary can choose exactly how long each message is delayed within the range $[0, \Delta]$. The lower bounds hold in an even stronger synchrony assumption: lockstep communication where communication proceeds in lockstep rounds.

We assume a Byzantine adversary that can corrupt up to f parties. We consider both static and strongly adaptive adversaries. A static Byzantine adversary must choose which parties to corrupt at the beginning of the protocol. 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 messages and send different messages instead. We assume a computationally unbounded adversary that can simulate other parties if required. When the adversary is computationally limited, we explicitly mention this.

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 3.1. 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, 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 [10, 23]. 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. A protocol solving Binary Crusader allowing m parties to disagree is called an all-but m Binary Crusader protocol, and is defined as follows:

Definition 3.2. 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 $y_i \in \{0, 1, \perp\}$. 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.** There exists some $y \in \{0, 1\}$ such that at most m nonfaulty parties i output $y_i \notin \{y, \perp\}$.
- **Termination.** If all nonfaulty parties participate in the protocol, they all complete it.

Note that an all-but 0 Binary Crusader Broadcast protocol is simply a Binary Crusader Broadcast protocol. 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 3.3. In a Binary Crusader Consensus protocol, every party i has an input $x_i \in \{0,1\}$. Every party outputs some value $y_i \in \{0,1,\bot\}$. 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, 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 [6]. 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 $\bot$. 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 $y \neq \bot$, then it received it in at least $n-f$ broadcasts. From the Correctness property, every other party either outputs y or $\bot$ in those broadcasts, and thus can output some y' such that $y' \notin \{y,\bot\}$ 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 $y' \notin \{y,\bot\}$, 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 [9], and from the subsequent work of Abraham *et al.* [1]. The lower bounds presented in theorem 4.1 and theorem 4.2 isolate a party in a similar way to the ones described in [9] and [1] respectively. However, unlike previous works, the lower bounds presented in this paper require the adversary to act in a Byzantine manner and actively simulate other parties. theorem 4.3 shows how previously known techniques can be used to isolate a large number of parties, and cause wide disagreement in the network. 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 4.1. *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 $P_0 \cup P_1 \cup \{i\}$ 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. Recall that as defined in section 3, a protocol is said to be p-correct if its properties hold with probability p or greater.

Theorem 4.2. *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 runs, 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, and the adversary will attempt to cause i to output 1 while other parties output 0. Informally, the adversary's strategy is to corrupt all parties that communicate with i (either by sending or receiving messages) throughout the run and delete all messages to i. The adversary then simulates i's responses to the messages it would have received, corrupts the parties that would have received those messages and causes them to act as if they received those messages. In addition, the adversary simulates a full run in W_1, and whenever a party sends a message to i in its simulation, it corrupts that party and causes it to send that message to i. This causes i to think it is in W_0 and all other parties to think they are in W_1, causing them to output different values.

More formally, 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 some party k, the adversary corrupts k. 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 k 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 4.3. *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 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 $n \geq 3f$.* [2]

[2]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.

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 runs, 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: $\forall i \in \{0,\dots,\ell-2\}$ $P_i = \{i \cdot m+1,\dots,(i+1)m\}$ and $P_{\ell-1} = \{n-m,\dots,n-1\}$. 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, $(\ell-2+1)m = (\lceil \frac{n-1}{m} \rceil - 1)m \le \frac{n-1}{m} \cdot m < n$, 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 $P_0,\dots,P_{\ell-2}$ and at most once more in $P_{\ell-1}$. Finally, the sets $P_0,\dots,P_{\ell-2}$ exactly contain the parties $1,\dots,(\ell-2+1)m$. Note that $(\ell-2+1)m = (\lceil \frac{n-1}{m} \rceil - 1)m \ge (\frac{n-1}{m}-1)m = n-1-m$, 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 $i \in \{0,\dots,\ell-1\}$ 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 $\sum_{j \in P_i} X_j \ge Y_i$ 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 $i \in \{0,\dots,\ell-1\}$, $\mathbb{E}[Y_i] > \epsilon f$. 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, $2\sum_{i=1}^{n} X_i \ge \sum_{i=0}^{\ell-1}\sum_{j \in P_i} X_j$.

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 4.2. 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

Blockchain system provided new revolutionary consensus protocols [13, 21] and with them came a new set of attacks [4, 11, 18]. One new style of attack focuses on the underlying peer-to-peer communication network and the ways it might affect the security of the system as a whole. These works suggest **Eclipse attacks** [15, 19], in which an adversary isolates a specific party (or group of parties), and causes it to fork off in ways that are economically advantageous to the attacker.

A natural question arises: are Eclipse-style attacks unique to the blockchain space or are they connected to more traditional attacks in the theory and literature on consensus? In consensus research, generic attacks on protocols are captured as lower bounds.

In this section, we make the conceptual connection between Eclipse-style attacks and theoretical lower bounds for Crusader broadcast. We show that many Eclipse-style attacks work because the underlying blockchain protocols are subquadratic and the protocol was not designed to take full power of forcing the adversary to be adaptive or to force the adversary to simulate. Hence Eclipse-style attacks can be viewed as specific attacks following the general lower bound for crusader broadcast, even with a mildly static adversary that cannot fully simulate as many other parties as it wants.

In the Eclipse attack, [15, 19] the adversary, by controlling a sufficient number of IP addresses, can monopolize all connections to and from a victim node. Once the node is isolated, the adversary can cause nodes to briefly locally confirm transactions that conflict with the majority of nodes. This is analogous to outputting different values in the attacks or our lower bound for crusader broadcast. Eclipse-style attacks may also combine selfish mining attacks [4, 11]. In this attack, the adversary filters communication to and from the isolated nodes and abuses the nodes' mining power to the adversary's advantage. This attack is also similar to the one described in theorem 4.3, 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.

We note that the difference between the attacks described in theorems 4.1 and 4.2 stems from the randomized nature of the communication graph, and not from the difference in the content of messages. The proof of theorem 4.1 would not change if a randomized Crusader Broadcast protocol uses a **static** communication graph. In addition, theorem 4.3 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. 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 means a weak protocol with near linear communication may allow a large adversary to partition the nonfaulty parties into two large groups that disagree on the output of the protocol.

Limitations of Real-World Adversaries

The attacks described 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 may not hold in some real-world systems. Since adversaries have limited compute power, 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 4.2 and 4.3 also needs 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 applicable to the real world?*

Surprisingly, the answer seems to be that they are applicable, as evidenced by previous works on eclipse attacks. We discuss how both limitations are overcome next.

Overcoming the need for strong adaptivity, due to protocol level flaws. As shown in [15, 19], both Bitcoin's and Ethereum's peer-to-peer communication protocols had flaws that allowed an adversary to easily monopolize a victim node's connections. When nodes restart, they initiate outgoing connections from tables storing addresses of known peers. The adversary fills those tables in advance with addresses of nodes controlled by the adversary and then causes the node to restart. After restarting, the node connects to peers from those tables, hence connecting to the adversary's nodes. Nodes may also 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 a malicious manner. Compare the Eclipse attack strategy above to our lower bounds for crusader broadcast. In this attack, the protocol flaw is such that the adversary does not need to be adaptive, let alone strongly adaptive. Even worse, the lower bound in theorem 4.1 only shows that there must exist 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. However, in the Eclipse attacks described in [15, 19], 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.

Overcoming the need for simulation due to the nature of proof of work. The second challenging assumption for a real-world attack is the assumption that the adversary has the power to simulate many honest parties. In our lower bounds for crusader broadcast, this 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, 25] for such examples). In order to fully simulate the behavior of the nonfaulty parties, an adversary needs to be able to break some of the cryptographic assumptions made in the design of the protocol. On the other hand, in current proof-of-work based blockchain systems, simulating honest parties "only" requires mining blocks with the correct information. The 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, Eclipse-style attacks suggest ways to mitigate this issue. For example, the adversary could conceivably utilize honest nodes to simulate the protocol for it. This can be done by 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.

To conclude, 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 real-world systems used today had flaws that didn't require the adversary to be so powerful in order to levy attacks.

Lessons from theory. Our lower bound suggests pricipled ways to design more secure protocols that will not allow adversaries with limited adaptivity and simulation power to succeed in their attacks. As suggested by [15, 19], measures could be taken in order to make it harder to fill the outgoing link tables with the adversary-controlled nodes' addresses. The proof of theorem 4.1 suggests that having a dynamically changing communication graph with outgoing edges being chosen randomly without much adversary control is the best long-term solution.

In addition, one could make it harder to simulate parts of the protocol. For example, by requiring more nodes to sign blocks, or by making more use of cryptography in the communication layer itself. This is indeed obtained using BFT-based finality gadgets [5].

6 SUBQUADRATIC CRUSADER BROADCAST PROTOCOL

After focusing on lower bounds for Crusader Broadcast, in this section, we explore upper bounds. We start with a trivial information theoretic crusader broadcast protocol with $O(n^2)$ communication. Our lower bound proves that this folklore construction is in fact tight for an unconditional adversary.

We then explore how using randomization and assuming a PKI can circumvent the $\Omega(n^2)$ lower bound for crusader broadcast. In the second protocol, the $O(n^2)$ all-to-all cost is replaced by a gossip procedure [22]. This lowers the overall communication cost to $O(n \cdot \text{polylog} n)$ at the cost of increasing the round complexity. The gossip protocol heavily relies on randomization to limit the adversary's ability to guess the communication pattern. We analyze this protocol against static adversaries. 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.

We start with a simple construction of a Crusader Broadcast protocol resilient to f strongly adaptive and computationally unbounded Byzantine corruptions, as long as $n > 3f$.

Theorem 6.1. *The* $\mathsf{IT-CrusaderBroadcast}$ *protocol is a Crusader Broadcast protocol resilient to* f *strongly adaptive, computationally unbounded Byzantine corruptions in a synchronous system if* $n > 3f$.

Proof. Each property is proven individually.

Validity. Assume the sender s is nonfaulty with input x. In the beginning of the protocol it sends the message $\langle \text{"sender"}, x, \rangle$ to all parties. Every nonfaulty party receives that message up to Δ time after that, and sends $\langle \text{"forward"}, x \rangle$

Algorithm 1 $\mathsf{IT-CrusaderBroadcast}_i$

```
1:  if $i$ is the sender $s$ with input $x$ then
2:      send the message $\langle$"sender"$, x\rangle$ to all parties
3:  wait $\Delta$ time
4:  if a single $\langle$"sender"$, m\rangle$ message was received from $s$ while waiting then
5:      send $\langle$"forward"$, m\rangle$ to all parties
6:  wait $\Delta$ time
7:  if there exists a value $m$ for which $\langle$"forward"$, m\rangle$ was received from at least $n - f$ parties then
8:      output $m$ and terminate
9:  else
10:     output $\bot$ and terminate
```

to all parties. After Δ time, every nonfaulty party will receive a $\langle$"forward"$, x\rangle$ message from every nonfaulty party. Since there are $n - f$ nonfaulty parties, every nonfaulty party then outputs x.

Correctness. Observe two nonfaulty parties i, j that output two non-$\bot$ values m_i, m_j respectively. This means that i received the message $\langle$"forward"$, m_i\rangle$ from at least $n - f$ parties, and j received the message $\langle$"forward"$, m_j\rangle$ from at least $n - f$ parties. By assumption, $n > 3f$, and thus $2(n - f) = 2n - 2f = n + (n - 2f) \geq n + f + 1$. Therefore, i and j received the aforementioned messages from at least $f + 1$ common parties. At least one of those parties must be nonfaulty, and nonfaulty parties only send a single "forward" message to all parties. Therefore, it must be the case that $m_i = m_j$.

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

Note that in the above protocol, the sender sends $O(n)$ messages, and each nonfaulty party sends $O(n)$ messages as well. This results in a protocol with $O(n^2)$ message complexity, showing that the lower bound above is tight.

The folklore $O(n^2)$ protocol, presented in algorithm 2, 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 $\bot$ instead. This protocol is captured in algorithm 2. 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 6.2. *The* $\mathsf{CrusaderBroadcast}$ *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.

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

Algorithm 2 $\mathsf{CrusaderBroadcast}_i$

```
1: $val \leftarrow \bot$
2: if $i$ is the sender $s$ with input $x$ then
3:     $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, x)$
4:     send the message $\langle$"sender", $x, \sigma\rangle$ to all parties
5: wait $\Delta$ 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:     $val \leftarrow m$
8:     send $\langle$"forward", $m, \sigma\rangle$ to all parties
9: wait $\Delta$ 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:     $val \leftarrow \bot$
12: output val and terminate
```

$\langle$"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 $\bot$. 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-$\bot$ 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-$\bot$ values, so $val_i \neq \bot$ and $val_j \neq \bot$. Party i only updates val_i to $m_i \neq \bot$ 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 $\bot$. Finally, j outputs $val_j = \bot$, contradicting the fact that it output some value $m_j \neq \bot$.

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

This simple protocol is based on the fact that for correctness to hold, it is enough to make sure that any value heard by a nonfaulty party needs to be heard by all nonfaulty parties (or at least the fact that two nonfaulty parties heard different values). In order to reduce the communication costs of the protocol, it is possible to replace the expensive all-to-all communication round with a more efficient *gossip* procedure. Using well-known results about gossip [7, 16, 22], we know that parties can exchange information between them by proceeding in rounds and in each round choosing a party to divulge all heard information to. Using this technique, it is guaranteed that in $O(\log n)$ rounds all parties will hear all nonfaulty parties' initial information. When dealing with a constant number of Byzantine faults, simply raising the number of parties with which each party communicates in each round yields the same analysis, showing that such a protocol requires $O(n \cdot \mathrm{polylog}(n))$ messages to be sent overall, yielding a subquadratic Crusader Broadcast protocol. It is also possible to reduce the size of the messages by parties sending up to 2 of the values they heard up until that point. This is enough to detect equivocation, while guaranteeing that message size remains constant. Note that an adaptive adversary can make sure that no nonfaulty party receives a message m from an informed party i in the first rounds of the protocol by corrupting the parties which received messages from i, requiring more rounds and more overall communication. This shows that a subquadratic randomized protocol exists which is resilient to non-adaptive adversaries, but stops working when the adversary can corrupt parties adaptively.

Another approach for breaking the quadratic message barrier is by relying on stronger cryptographic primitives. This has been useful in reducing the communication costs of protocols solving related tasks such as consensus [20, 25]. For example, if $n > 3f$ it is possible to use threshold cryptography. Given a well known threshold k, a threshold signature scheme allows parties to sign a message individually, and then compressing k such signatures into a single collective signature, proving that at least k parties signed the message individually. Instead of having an all-to-all round as described in algorithm 2, parties that hear a value m from the sender can reply, sending a signature on m to the sender. Using a threshold signature scheme with a threshold of $2f+1$, the sender can combine those signatures into a single threshold signature proving that at least $2f+1$ parties replied with a signature on the value m. Since $2f+1$ parties constitute a Byzantine quorum, it is guaranteed that there is only one such threshold signature, meaning that if the sender then sends the threshold signature to all parties, they can safely output it. This technique, used in the non-equivocation round of the HotStuff protocol [25], yields a protocol with $O(n)$ communication costs and $O(1)$ rounds, but more heavily relies on the assumption that the adversary cannot simulate other parties.

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] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[6] 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.

[7] 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.

[8] Danny Dolev. The byzantine generals strike again. *Journal of algorithms*, 3(1):14–30, 1982.

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

[10] 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.

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

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

[13] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015.

[14] Vassos Hadzilacos and Joseph Y Halpern. Message-optimal protocols for byzantine agreement. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 309–323, 1991.

[15] 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.

[16] 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.

[17] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA '06*, 2006.

[18] 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.

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

[20] Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 32:1–32:16.

Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

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

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

[23] 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.

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

[25] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019.