Asynchronous Byzantine Agreement
in Incomplete Networks

Ye Wang
wangye@ethz.ch
ETH Zurich
Zurich, Switzerland

Roger Wattenhofer
wattenhofer@ethz.ch
ETH Zurich
Zurich, Switzerland

ABSTRACT

The Byzantine agreement problem is considered to be a core problem in distributed systems. For example, Byzantine agreement is often used to build a blockchain, a totally ordered log of records. Blockchains are asynchronous distributed systems, fault-tolerant against Byzantine nodes.

In the literature, the asynchronous byzantine agreement problem is studied in a fully connected network model where every node can directly send messages to every other node. This assumption is questionable in many real-world environments. In the reality, nodes might need to communicate by means of an incomplete network, and Byzantine nodes might not forward messages. Furthermore, Byzantine nodes might not behave correctly and, for example, corrupt messages. Therefore, in order to truly understand Byzantine Agreement, we need both ingredients: asynchrony and incomplete communication networks.

In this paper, we study the asynchronous Byzantine agreement problem in incomplete networks. A classic result by Danny Dolev proved that in a distributed system with n nodes in the presence of f Byzantine nodes, the vertex connectivity of the system communication graph should be at least $(2f+1)$. While Dolev's result was for synchronous deterministic systems, we demonstrate that the same bound also holds for asynchronous randomized systems. We show that the bound is tight by presenting a randomized algorithm, and a matching lower bound. This algorithm is based on a protocol which allows other Byzantine agreement algorithms to be implemented in incomplete networks.

Index Term— Blockchain, Byzantine agreement, communication network, randomized algorithms

1 INTRODUCTION

Byzantine agreement is at the heart of understanding distributed systems. Most existing work about byzantine agreement assumes a fully connected network, i.e., every node in the distributed system can directly communicate with every other node. In reality, however, nodes are often connected by an unreliable network, such as the Internet. To communicate, two nodes must exchange messages, and these messages will be forwarded by relay nodes, which are controlled by third parties. Relay nodes may be compromised, even Byzantine. A relay node may for instance decide to corrupt or simply drop messages. This is particularly true in world-scale distributed systems, and world-scale systems (blockchains) are predominantly responsible for the current reawakened interest in byzantine agreement.

The Byzantine agreement problem has also been studied *in a network*. Already as early as 1982, Danny Dolev [7] showed that two conditions are both necessary and sufficient to achieve Byzantine agreement in an n-node system: As usual, we need the number of Byzantine nodes f to obey $f < \frac{n}{3}$; In addition, the vertex connectivity of the communication graph cannot be less than $(2f+1)$.

Dolev's fundamental work had only considered the synchronous model, i.e., all communication happened in synchronous rounds. Moreover, the result was restricted to deterministic algorithms.

In world-scale systems, it is difficult to argue for synchronous communication. Consequentially, the focus has shifted away from synchronous systems towards asynchronous systems. Even though some form of synchrony is usually needed for liveness, safety is guaranteed even in completely asynchronous systems. This is the case for permissionless blockchain systems such as Bitcoin [17], and also for permissioned systems such as PBFT [6].

In this paper, we study the effect of *incomplete networks* on byzantine agreement in *asynchronous* distributed systems. We have a system with n nodes, each with a binary input. At most f nodes, with $f < \frac{n}{3}$, are Byzantine. The nodes are interconnected by an incomplete network, which can be represented by an undirected graph $G = (V_G, E_G)$. An example of such an incomplete network is shown in Figure 1.

It was proven in [10] that there is no deterministic algorithm which can solve Byzantine agreement in an asynchronous distributed system, with $f = 1$, i.e., a single Byzantine node, or even a crash node. However, randomization helps [2], and consequently our algorithm will be randomized as well. More precisely, we present a randomized algorithm that solves the Byzantine agreement problem in asynchronous distributed systems with high probability, as long as the vertex connectivity of the network G is at least $(2f+1)$. We also show that this bound is tight by proving that there is no randomized algorithm which can solve the asynchronous Byzantine agreement problem when the vertex connectivity of G is less than $(2f+1)$. In other words, we demonstrate that with

the help of randomization, Dolev's original bounds [7] also hold in asynchronous networks.

The rest of this paper is organized as follows: In the next section, we give an overview of related work. Section 3 defines the model that we study in this paper. In Section 4, we present a randomized algorithm for solving the Byzantine agreement in our model. We give the necessary condition for solving the Byzantine agreement problem under the model in Section 5. We conclude our work in Section 6.

2 RELATED WORK

In this section, we discuss all the known related work on Byzantine agreement which are studied in asynchronous distributed systems and incomplete networks.

The Byzantine agreement problem is essential in distributed systems. It was proved in [10] that there is no deterministic algorithm that can solve the Byzantine agreement problem in the presence of even a single Byzantine node. Later Lamport et al. [13] proved that if the number of Byzantine nodes is larger than or equal to $\frac{n}{3}$, it is impossible for any algorithm to solve the Byzantine agreement problem with n nodes. The first randomized algorithm was proposed by Ben-Or [2]. With this algorithm, all correct nodes will decide on the same output with high probability even when a constant fraction of nodes is faulty. Bracha [3] improved the previous result and proposed a randomized algorithm that solves the asynchronous Byzantine agreement problem if less than one third of nodes are Byzantine.

The connectivity conditions of communication graphs have been studied right when the topic emerged. Danny Dolev [7] studied Byzantine agreement in synchronous distributed systems among n nodes in the presence of f Byzantine nodes and figured out that it is possible to achieve Byzantine agreement in networks with vertex connectivity at least $(2f + 1)$ and $n \geq 3f + 1$. Later Dolev et al. and Abraham et al. [1, 8] showed that the approximate Byzantine agreement problem can be solved in synchronous networks if and only if $(2f + 1)$ vertex connectivity is given.

Recently, there are a series of results under the local broadcast model. In contrast to the orthodox point-to-point communication model, under the local broadcast model, all neighbors of a transmitting node are guaranteed to receive identical messages. A lower connectivity requirement under the local broadcast model in the synchronous setting was obtained in [11]. In the presence of f Byzantine nodes, the following conditions are both necessary and sufficient. The communication graph G with n nodes has minimum degree $2f$ and G is $\left(\left\lfloor \frac{3f}{2} + 1 \right\rfloor\right)$-connected. The local broadcast model in the asynchronous distributed system was considered in [18]. They show that it is necessary to have $(2f + 1)$ vertex connectivity for solving the approximate Byzantine agreement problem. This bound keeps the same as the bound of the point-to-point communication model given in [8].

We are the first to study the exact Byzantine agreement problem in asynchronous distributed systems, as the comparison between our work and related work shown in Figure 2. In contrast to results in [8, 18], which consider approximate Byzantine agreement, we study the exact Byzantine agreement problem. To obtain an approximate Byzantine agreement, nodes are only required to obtain values

that are close to each other, rather than identical. However, in many real-world systems, such as financial systems, one requires precise values. Any minor deviation may accumulate and cause serious consequences. Other work [7, 11] solved exact Byzantine agreement solely in synchronous distributed systems by deterministic algorithms. Unfortunately, assuming synchrony is often not realistic in many world-scale systems. The impossibility for deterministic algorithms to solve the asynchronous Byzantine agreement problem inspires us to propose a randomized algorithm that achieves Byzantine agreement on undirected incomplete $(2f + 1)$-connected communication graphs in asynchronous distributed systems with n nodes in the presence f Byzantine nodes.

3 MODEL AND NOTATION

3.1 Distributed System

Given a distributed system with n nodes, the communication network between nodes is represented by an undirected graph $G = (V_G, E_G)$. Nodes u and v can send messages to each other if and only if they are adjacent in G, i.e., $(u, v) \in E_G$. We will use the terms of *node* and *vertex* interchangeably in this paper. The communication channels between nodes are authenticated. Nodes can recognize who is the sender of the message when they receive a message. Messages cannot be modified by any third party if messages are delivered via the authenticated channel between adjacent nodes. However, relay nodes on paths between nodes have the ability to modify messages or even generate fake messages.

3.2 System parameters

The system we study in this paper has two critical parameters.

Asynchronous systems or synchronous systems: In a synchronous system, there is an upper bound on the message delivery delay from one node to another which is known by all nodes. Nodes take actions in rounds. Each round takes a constant period of time which is sufficient for nodes to send messages, do local computations and accept incoming messages. In asynchronous distributed systems, the message delay from one node to another has no finite upper bound. Messages may be delayed for arbitrary time periods. However, messages will eventually be delivered. Nodes take actions when they are activated by events, such as messages arriving. In this paper, we study asynchronous systems.

Broadcast transmission or point-to-point transmission: if the transmission mechanism is point-to-point, nodes can send a message to at most one neighbor at a time. If the transmission mechanism is broadcast, node u sends identical messages to all of its neighbors at the same time. In this paper we study the point-to-point transmission mechanism.

3.3 Byzantine agreement

There are Byzantine nodes in distributed systems, which may behave arbitrarily. A Byzantine node may corrupt or simply drop messages. The system requires agreement among all nodes in the system, e.g. a total order of blocks in blockchain systems. Because there are general reduction protocols from multivalued agreement to binary agreement [16, 19], we focus on the Byzantine binary agreement problem in this paper.
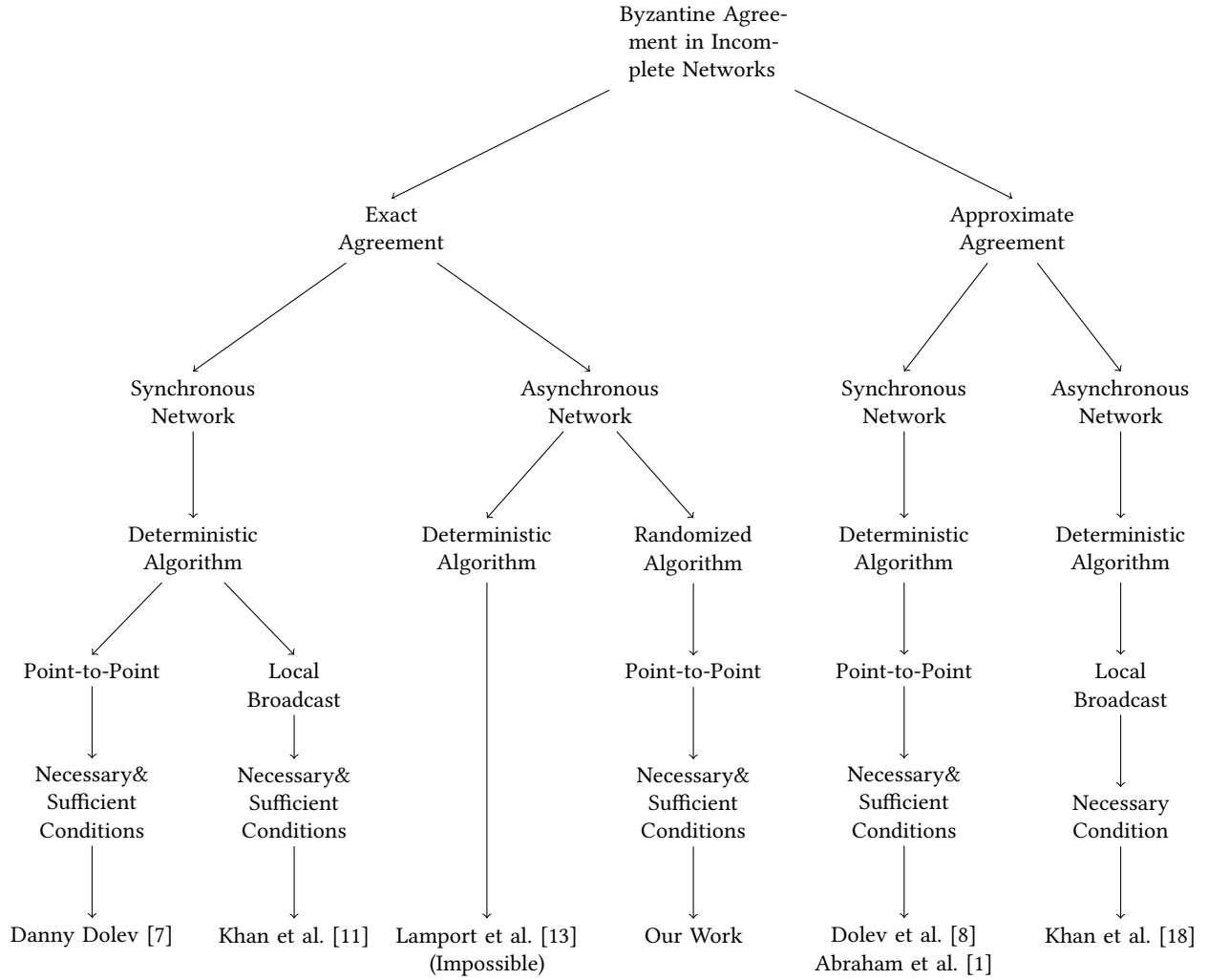
Byzantine Agree-
ment in Incom-
plete Networks

Exact
Agreement

Approximate
Agreement

Synchronous
Network

Asynchronous
Network

Synchronous
Network

Asynchronous
Network

Deterministic
Algorithm

Deterministic
Algorithm

Randomized
Algorithm

Deterministic
Algorithm

Deterministic
Algorithm

Point-to-Point

Local
Broadcast

Point-to-Point

Point-to-Point

Local
Broadcast

Necessary&
Sufficient
Conditions

Necessary&
Sufficient
Conditions

Necessary&
Sufficient
Conditions

Necessary&
Sufficient
Conditions

Necessary
Condition

Danny Dolev [7]

Khan et al. [11]

Lamport et al. [13]
(Impossible)

Our Work

Dolev et al. [8]
Abraham et al. [1]

Khan et al. [18]

Figure 1: Comparison with previous work.

Every node has a binary input, and we want correct nodes decide on a binary value which satisfies the following conditions,

Agreement (exact): all correct nodes decide for the same value.

Agreement (approximate): for any preassigned $\epsilon > 0$, all correct nodes decide with outputs that are within ϵ of each other.

Termination (deterministic): all correct nodes terminate in a finite time.

Termination (probabilistic): the probability that a correct node is undecided after r steps approaches 0 as r approaches infinity.

Validity: the decision value must be the input value of a node.

In contrast to the exact agreement, the approximate agreement is not valid for many real-world distributed systems. Thus, we focus on the exact agreement problem in this paper.

Because it is impossible for any deterministic algorithm to solve exact Byzantine agreement problem in asynchronous distributed systems in the presence of a single faulty node, we study randomized algorithms with the probabilistic termination condition. There exists a source that generates random numbers in randomized

algorithms. Nodes have access to these random numbers during executions. The agreement made by nodes is based on their input and these random values. Therefore, with the same input, the behavior of nodes can be different because of these random numbers.

3.4 Connectivity

A **path** P in an undirected graph $G = (V_G, E_G)$ is a finite sequence of edges which joins a sequence of distinct vertices. A uv-path P_{uv} is a path between nodes u and v. Nodes u and v are endpoints of P_{uv}. Nodes other than u and v in P_{uv} are internal nodes of P_{uv}.

If there is a path between every node pair $u, v \in V_G$, then G is connected. G is k-connected if G is still connected after removing $k - 1$ arbitrary nodes. There is a classic result for k-connected graphs:

THEOREM 1 (MENGER'S THEOREM [14]). *An undirected graph $G = (V_G, E_G)$ is k-connected if and only if for any two nodes $u, v \in V_G$,*

there are k node disjoint uv-paths. Two uv-paths are disjoint if and only if they do not have any identical internal node.

4 SUFFICIENT CONDITIONS

In this section, we present an algorithm that solves the asynchronous Byzantine agreement problem with high probability on $G = (V_G, E_G)$, when G is $(2f + 1)$-connected. We will introduce this algorithm in three levels, top to bottom. Lower level protocols provide fundamental functions for constructing higher level protocols.

The primary protocol is discussed in Subsection 4.1, which is the top level of our algorithm. It is an f-resilient agreement protocol that describes the behavior of nodes, including how nodes communicate with each other and how they decide on the agreement.

The middle level of our algorithm is a broadcast algorithm, as the primary f-resilient agreement protocol above requires a broadcast communication mechanism. However, our model is built on the point-to-point communication mechanism, which does not meet the requirement. We use an authenticated double-echo broadcast algorithm to fix this gap. We discuss this broadcast algorithm in Subsection 4.2. With this broadcast algorithm, identical messages will be delivered to all nodes.

The bottom level of our algorithm is an algorithm that allows any node pair to communicate with each other in incomplete networks. The middle level algorithm, i.e., the authenticated double-echo algorithm is implemented in systems where nodes can directly communicate with each other. Therefore, we design an asynchronous purifying algorithm to effectuate the broadcast algorithm in our model. In Subsection 4.3, Subsection 4.4 and Subsection 4.5, we present the asynchronous purifying algorithm, which ensures that the message delivery between any node pair is correct in an incomplete communication network with the presence of Byzantine nodes. We explain how nodes send messages, transmit messages as internal nodes on a path and accept messages respectively.

4.1 Byzantine agreement Algorithm

The top level protocol is an f-resilient agreement protocol in asynchronous systems in the presence of $f < \frac{n}{3}$ Byzantine nodes, which is derived from the probabilistic protocol in [3]. In asynchronous systems, there is no global real-time clock. Nodes take actions when they receive messages from other nodes.

Nodes take actions in phases. We present the protocol of phase i in Algorithm 1. The algorithm starts in phase 0. The initial value of v_u is the input of node u and $decision_u$ is initialized as $\bot$. We assume that nodes communicate with each other by reliable broadcast channels in this subsection. If a message is delivered to one node, it is also delivered to other nodes in the system.

In each phase, node u takes actions in three rounds. Node u keeps a value at the beginning of each phase and broadcasts this value to other nodes in the first round. After validating $(n - f)$ messages of the first round from other nodes, node u sets its value as the majority of these $(n-f)$ messages and enters in the second round. In the second round, node u broadcasts the new value and waits until validating $(n - f)$ messages of the second round from other nodes. If there is a value which is accepted in more than $\frac{n}{2}$ messages, then u sends this value at the beginning of the third round. Otherwise, node u sends an empty message in the third round. Node u waits

---

Algorithm 1 Code for node u, $Phase(i)$, $i = 0, 1, ...$

1: **round 1**
2:     **broadcast**(source = u, round = $3i + 1$, value = v_u)
3:     **wait** until validate $(n - f)$ messages of round $3i + 1$
4:     **if** more than $\frac{n-f}{2}$ messages of round $3i + 1$
5:     have the same value v_m **then**
6:         $v_u \leftarrow v_m$
7:     **end if**
8: **end round 1**
9:
10: **round 2**
11:     **broadcast** (source = u, round = $3i + 2$, value = v_u)
12:     **wait** until validate $(n - f)$ messages of round $3i + 2$
13:     **if** more than $\frac{n}{2}$ messages have the same value v_m
14:     other than v_u **then**
15:         $v_u \leftarrow v_m$
16:         $ready \leftarrow True$
17:     **else**
18:         $ready \leftarrow False$
19:     **end if**
20: **end round 2**
21:
22: **round 3**
23:     **if** $ready$ **then**
24:         **broadcast** (source = u, round = $3i + 3$,
25:                 value = v_u)
26:     **else**
27:         **broadcast** (source = u, round = $3i + 3$,
28:                 value = $\emptyset$)
29:     **end if**
30:     **wait** until validate $(n - f)$ messages of round $3i + 3$
31:     **if** more than $2f$ messages have the same value
32:     $v_m \neq \emptyset$ **then**
33:         $v_u \leftarrow v_m$
34:         $decision_u \leftarrow v_u$
35:         **for** $j \leftarrow 1$ to 3 **do**
36:             **broadcast** (source = u, round = $3(i + 1) + j$,
37:                     value = v_u)
38:         **end for**
39:         **terminate**
40:     **else**
41:         **if** more than f messages have the same value
42:         v_m **then**
43:             $v_u \leftarrow v_m$
44:         **else**
45:             $v_u \leftarrow coin\_toss$ (0 or 1 with probability $\frac{1}{2}$)
46:         **end if**
47:     **end if**
48: **end round 3**
49:
50: **go to** $Phase(i + 1)$

---

until validating $(n - f)$ messages of the third round from other nodes. If more than $2f$ messages have the same value, then u makes the decision. Node u terminates the algorithm when it decides on

$decision_u$. Before termination, it broadcasts three messages for the next phase. If less than $2f$ messages but more than f messages have the same value, then u does not change v_u and keeps v_u in the next phase. If there is no value exists in more than f messages, node u takes a coin toss to get a random value and keeps this value in the next phase.

This protocol solves the Byzantine agreement problem because it satisfies validity, exact agreement, and probabilistic termination. We discuss the correctness of this protocol in these three aspects.

THEOREM 2. *Algorithm 1 solves the Byzantine agreement problem in asynchronous distributed systems in the presence of $f < \frac{n}{3}$ Byzantine nodes.*

LEMMA 1. *Algorithm 1 satisfies the validity property.*

PROOF. There are two possible situations of node inputs in the system. In the first situation, all correct nodes have the same input value. In the second situation, correct nodes have different input values.

If all correct nodes have the same input v, then all of them will receive more than $\frac{n-f}{2} > f$ messages of round 1 with the identical value v. In round 2, v_u does not change because Byzantine nodes cannot create more than $\frac{n}{2}$ copies of malicious messages. For the same reason, all correct nodes will decide on v in round 3.

If nodes have different inputs, no matter which value they agree on, the validity property is always satisfied. □

LEMMA 2. *Algorithm 1 satisfies the exact agreement property.*

PROOF. First, we claim that two correct nodes u and w won't decide on different values in round $3k + 3$. Suppose not: then node u decides on 0 while node w decides on 1 in round $3k + 3$. Because both of them validate more than $2f$ messages, there are two correct nodes u', and w' get ready with value 0 and value 1 in round $3k+2$, i.e., $ready = TRUE$, respectively. Node u' validates more than $\frac{n}{2}$ messages with value 0 in round $3k + 2$ and node u' validates more than $\frac{n}{2}$ messages with value 1 in round $3k+2$. There must be a node broadcasting two messages with value 0 and value 1 separately in round $3k + 2$, which is impossible. Thus, correct nodes decide on the same value in the same round.

Suppose that node u decide on 0 in round $3k + 3$, it validates $(2f + 1)$ messages with value 0. Hence, other correct nodes validate at least $(f + 1)$ messages of these $(2f + 1)$ messages which are validated by u. Because node u will continue broadcasting correct messages with value 0 in phase $k + 1$. All correct nodes have the same value from round $3(k + 1) + 1$ and all of them will decide on 0 in round $3(k + 1) + 3$.

□

LEMMA 3. *Algorithm 1 satisfies the probabilistic termination property.*

PROOF. Let us consider node actions in phase k. All nodes are still active in phase k. There are four possible situations that a correct node u can be found in round $3k + 3$.

In the first situation, if node u validates more than $2f$ messages with value v in round $3k+3$, apparently, all correct nodes will decide on value v in round $3(k + 1) + 3$ with probability 1.

In the second situation, node u validates more than f messages with value v in round $3k + 3$. Other correct nodes will not decide on the value $v' \neq v$ or directly set its value to v'. Otherwise, with the same argument in the proof of Lemma 2, there must be a node broadcasting two messages with value 0 and value 1 respectively in round $3k + 2$, which is impossible. Thus, the probability that other correct nodes start with value v in phase $k + 1$ is greater than $2^{-(n-f)}$.

In the third situation, node u validates less than f messages with value v in round $3k + 3$. Other correct nodes might validate v but won't validate a value other than v. The probability that all correct nodes start with value v in phase $k + 1$ is greater than $2^{-(n-f)}$.

In the fourth situation, node u has not validated a message with value v. Other correct nodes have not validated more than f messages with value v. Otherwise, node u will validate v at least once. Therefore, all correct nodes toss coins and the probability that all correct nodes start with value v in phase $k + 1$ is $2^{-(n-f)}$.

In all of these situations, algorithms terminates in phase $k+1$ with probability greater than or equal to $2^{-(n-f)}$. Thus, the probability that the algorithm never terminates is $\mathbb{P}(never\ terminating) \leq \lim_{k \to \infty}(1 - 2^{-(n-f)})^k = 0$.

□

4.2 Byzantine Reliable Broadcast

In our model, we assume that the transmission mechanism between nodes is point-to-point, which does not meet the requirement of the top level protocol in Subsection *4.1*; we use multiple **broadcast(·)** in Algorithm 1. We seek for a reliable broadcast to fix this gap. In this subsection, we discuss the details of a Byzantine reliable broadcast algorithm that helps us to implement the algorithm in our model.

The Byzantine reliable broadcast allows nodes to broadcast in identical message m to all nodes in the system. Nodes will validate the message as the response of the broadcast. A Byzantine reliable broadcast algorithm satisfies five properties [5],

Validity: if a correct node u broadcasts a message m, then every correct node eventually validates m.

No duplication: every correct node validates message m at most once in the broadcast of message m.

Integrity: if a correct node w validates a message m from another correct node u, then m was broadcast by u before w validating m.

Consistency: if a correct node w validates a message m and another correct node w' validates a message m' in the same round, then $m = m'$.

Totality: if a correct node w validates a message m, then other correct nodes eventually validate m.

We follow the authenticated double echo broadcast algorithm in [3]. The algorithm consists of three parts: broadcasting messages, echoing messages, and validating messages, which are shown in Algorithm 2, Algorithm 3, and Algorithm 4, respectively. Note that we assume full communication capability among nodes in this subsection, but it is not true in our model, we will explain how to solve this problem in Subsection *4.3*, Subsection *4.4* and Subsection *4.5*.

Algorithm 2 indicates how node u broadcasts a message m. It sends m to all other nodes in the system with a *initial* label.

Algorithm 2 Broadcast Message m = (source = u, round = k, value = v_u), Code for node u

---

1: **Send** $(m, \text{from} = u, \text{label} = initial)$ to all nodes in the system.

Algorithm 3 Echo Message m = (source = u, round = k, value = v_u), Code for node w

---

1: $echo_w \leftarrow \perp^N$
2: $ready_w \leftarrow \perp^N$
3: **upon** accept message $(m, \text{FROM} = u, \text{LABEL} = initial)$
4:     **send** $(m, \text{from} = w, \text{label} = echo)$ to other nodes.
5: **end upon**
6:
7: **upon** accept message $(m = (\text{SOURCE} = u, \text{ROUND} = k, \text{VALUE} = v_u), \text{FROM} = v, \text{LABEL} = echo)$
8:     **if** $(\text{source} = u, \text{round} = k, ...) \notin echo_w[v]$ **then**
9:         $echo_w[v] \leftarrow echo_w[v] \cup m$
10:     **end if**
11: **end upon**
12:
13: **upon** $\#(m \in echo_w[p], \forall p \in V_G) > \frac{n+f}{2}$ and $m \neq \perp$
14:     **if** have not sent $(m, \text{from} = w, \text{label} = ready)$ **then**
15:         **send** $(m, \text{from} = w, \text{label} = ready)$
16:         to other nodes.
17:     **end if**
18: **end upon**
19:
20: **upon** accept message $(m = (\text{SOURCE} = u, \text{ROUND} = k, \text{VALUE} = v_u), \text{FROM} = v, \text{LABEL} = ready)$
21:     **if** $(\text{source} = u, \text{round} = k, ...) \notin ready_w[v]$ **then**
22:         $ready_w[v] \leftarrow ready_w[v] \cup m$
23:     **end if**
24: **end upon**
25:
26: **upon** $\#(m \in ready_w[p], \forall p \in V_G) > f$ and $m \neq \perp$
27:     **if** have not sent $(m, \text{from} = w, \text{label} = ready)$ **then**
28:         **send** $(m, \text{from} = w, \text{label} = ready)$
29:         to other nodes.
30:     **end if**
31: **end upon**

The algorithm is called authenticated double-echo broadcast because it has two echo steps, which are indicated in Algorithm 3. In the first step, node w sends message m with *echo* label to other nodes when it accepts the message m with *initial* label from u. In the second step, w sends message m with *ready* label to other nodes when it accepts more than $\frac{n+f}{2}$ copies of the message m with *echo* label from other nodes or when it accepts more than f copies of the message m with *ready* label from other nodes.

We introduce the validating step in Algorithm 4. The message m is validated by node w when it accepts more than $2f$ copies of the message m with *ready* label from other nodes. This message is stored in the set Val. Set Val is an empty set at the beginning of the algorithm.

Algorithm 4 Validate Message m = (source = u, round = k, value = v_u), Code for node w

---

1: $Val \leftarrow \emptyset$
2: **upon** $\#(ready_w[p] = m, \forall p \in V_G) > 2f$ **AND** $m \neq \perp$
3:     **if** $(\text{source} = u, \text{round} = k, ...) \notin Val$ **then**
4:         **validate** m as a message broadcast by u
5:         in round k
6:     $Val \leftarrow Val \cup m$
7:     **end if**
8: **end upon**

THEOREM 3. *The authenticated double-echo broadcast algorithm is a Byzantine reliable broadcast.*

PROOF. We prove Theorem 3 by indicating that the authenticated double-echo broadcast algorithm satisfies five properties of the Byzantine reliable broadcast in Lemma 4, Lemma 5, Lemma 6, Lemma 7 and Lemma 8. □

LEMMA 4. *The authenticated double-echo broadcast algorithm satisfies the validity property.*

PROOF. If a correct node u broadcasts a message m, then other correct nodes will accept it and send a *echo* message to others nodes in the system. Because $f < \frac{n}{3}$, we have $n - f > \frac{n+f}{2}$ and $f < \frac{n+f}{2}$. Correct nodes will accept at least $\frac{n+f}{2} + 1$ *echo* copies of message m and send a *ready* message to others.

For the same reason, correct nodes will eventually accept at least $(2f + 1)$ copies of message m with *ready* label from other correct nodes. Hence, message m will be validated by every correct node eventually. □

LEMMA 5. *The authenticated double-echo broadcast algorithm satisfies the no duplication property.*

PROOF. The no duplication property is intuitive because m is validated by node w if it is not in the set Val. m will be added to Val after validation by node w. Hence every correct node only validates m once. □

LEMMA 6. *The authenticated double-echo broadcast algorithm satisfies the integrity property.*

PROOF. If a correct node sends a message m with *ready* label to other nodes, it needs at least $\frac{n+f}{2} + 1 > f$ accepted *echo* messages or $(f + 1)$ accepted *ready* messages. Byzantine nodes have no ability to realize these two events.

A correct node needs at least $(2f + 1)$ accepted *ready* copies of a message m to validate m. It is impossible for Byzantine nodes to create $(2f + 1)$ fake messages with *ready* label because there are less than f Byzantine nodes in the system. Thus, message m was broadcast from another node previously. □

LEMMA 7. *The authenticated double-echo broadcast algorithm satisfies the consistency property.*

PROOF. The integrity property of the authenticated double-echo broadcast algorithm implies that m and m' are all broadcast by node u previously. Because node u only broadcasts one message in

a round, we must have $m = m'$. This message was broadcast by u in this broadcast round. □

LEMMA 8. *The authenticated double-echo broadcast algorithm satisfies the totality property.*

PROOF. If a correct node u validates a message m, it has already accepted at least $(2f + 1)$ messages with *ready* label. At least $(f + 1)$ of them are sent by correct nodes.

Thus, every correct node will accept at least $(f + 1)$ messages with a *ready* label from other correct nodes eventually. Because the authenticated double-echo broadcast algorithm requires a node to send a message of m with *ready* label when it accepts $(f + 1)$ messages with *ready* label.

For this reason, every correct node will send a *ready* message of m to each other. Every correct node will validate m eventually. □

4.3 Send Messages

The authenticated double-echo broadcast algorithm was initially implemented in the system where nodes can communicate with each other directly. However, in our model, communication is restricted because the network is incomplete. The authenticated channels are only between adjacent nodes of the network. Danny Dolev introduced the Purifying algorithm, which has been used to solve communication problems on incomplete graphs [7]. However, this algorithm only works in synchronous systems.

In this subsection, Subsection *4.4* and Subsection *4.5*, we proposed an asynchronous purifying algorithm that fixes the communication gap in the system where not every pair of nodes can send and receive messages directly. The algorithm has the following properties such that the correctness of the authenticated double-echo broadcast algorithm is guaranteed.

Validity: if a correct node u sends a message m to another correct node w, then node w eventually accepts m.

No duplication: if a correct node u sends a message m to another correct node w, then node w only accepts m once.

Integrity: if a correct node w accepts a message m from another correct node u, then m was broadcast by node u before node w accepting message m.

We introduce our algorithm by giving an example of message delivery, i.e., node u sends a message m to node w while node v is an internal node of a path between u and w. The asynchronous purifying algorithm consists of three parts: sending m from the starting node u, transmitting m via a path by an intermediate node v, and accepting m at the destination node w, these steps are introduced respectively in this subsection, Subsection *4.4* and Subsection *4.5*.

Algorithm 5 demonstrates how node u acts when u decides to send a message m to node w. As we discussed in Subsection *4.2*, node u sends a message m with a label. When node u sends a message m to its neighbors, it sends the message m together with the label, its own identifier u, and the message flooding path. Initially, the flooding path of the message is empty, which is represented by $\perp$.

4.4 Transmit Messages

Because the communication graph is not complete, messages cannot be directly delivered to the destination. Internal nodes on paths
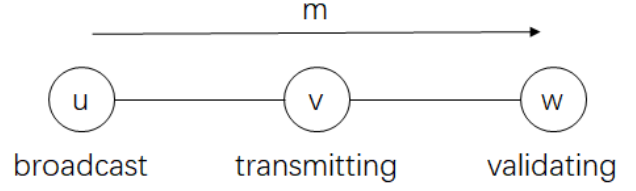

Figure 2: Node u sends a message m to node w via a path including node v.

Algorithm 5 Send $(m, \text{from} = u, \text{label} = l)$ to node w, Code for node u

---
1: **send** message $(m, \text{from} = u, \text{label} = l, \text{path} = \perp)$ to $u.neighbor$

---

between message source node and message destination node are responsible for transmitting messages. In our example, when node v receives a message m, from $= u$, label $= l$, path $= \Pi$ from a neighbor t, it will send this message to its neighbors other than t and store it in its local memory. The details are presented in Algorithm 6.

Node v first checks that if the message flooding path Π includes itself. If so, v discards this message. This step ensures that node v does not accept and send any duplicated message. Node v discards a message if the message flooding path is incorrect. If the message is not discarded, then node v stores this message, message label and the message flooding path in its local memory and sends (source $= u$, round $= k$, value $= v_u$, path $= \Pi - t$) to all neighbors except t.

Algorithm 6 Transmit $(m, \text{from} = u, \text{label} = l, \text{path} = \Pi)$, Code for node v

---
1: **upon** receive $(m, \text{FROM} = u, \text{LABEL} = l, \text{PATH} = \Pi)$ from t
2:    **if** $v \in \Pi$ **then**
3:       ignore the message
4:    **end if**
5:    **if** $u \notin \Pi$ **then**
6:       ignore the message
7:    **end if**
8:    **store** $(m, \text{from} = u, \text{label} = l, \text{path} = \Pi - t)$
9:       in memory list Mem
10:    **send** $(m, \text{from} = u, \text{label} = l, \text{path} = \Pi - t)$
11:       to $v.neighbor \setminus t$
12: **end upon**

4.5 Accept Messages

At node w, it accepts message m if it receives enough copies of m with the same label via disjoint flooding paths. The part of accepting messages is shown in Algorithm 7. Node w accepts a message m from node u with label l when it receives more than f identical copies with label l via disjoint flooding paths. This message is stored in $Acpt$. $Acpt$ is initialized as an empty set at the beginning of the whole algorithm.

Algorithm 7 Accept (m, from = u, label = l), Code for node w

1: $Acpt \leftarrow \emptyset$
2: **upon** #((m, FROM = u, LABEL = l, ...) $\in Mem$) > f **AND** their
   FLOODING PATHS are disjoint
3:     **if** (m, from = u, label = l) $\notin Acpt$ **then**
4:        **accept** m as a message from node u with label l
5:        $Acpt \leftarrow Acpt \cup (m, \text{from} = u, \text{label} = l)$
6:     **end if**
7: **end upon**

THEOREM 4. *The asynchronous purifying algorithm satisfies the validity, no duplication, and integrity properties, which guarantees the correctness of the authenticated double-echo broadcast algorithm in our model.*

PROOF. The no duplication property is intuitive because (m, from = u, label = l) is accepted if it is not in $Acpt$.

We prove the correctness of this asynchronous purifying algorithm from two perspectives. We fist prove the validity property in Lemma 9 by showing that every message m sent by node u will be accepted by node w eventually. Then we prove that if a message m is accepted by node w, it must be sent by node u first to verify the integrity property in Lemma 10.

□

LEMMA 9. *When a correct node u broadcasts a message m with label l, another correct node w will receive at least ($f + 1$) correct copies via disjoint flooding paths if the communication graph G has ($2f + 1$) vertex connectivity.*

PROOF. Because the connectivity of the communication graph G is ($2f + 1$), there are at least ($2f + 1$) disjoint paths between any node pair. Therefore, there are at least ($2f + 1$) disjoint paths between the source node u and the destination node w.

There are at most f byzantine nodes in the system. These nodes can only appear in at most f disjoint paths between u and w. Thus, there are at least ($f + 1$) disjoint paths only contains correct nodes between u and w.

Correct messages will be transmitted through these paths from node u to node w. Hence, node w will receive at least ($f + 1$) correct copies via disjoint flooding paths, and message m with label l from node u will be accepted by node w eventually. □

LEMMA 10. *Every message m with label l which is validated in the asynchronous purifying algorithm is a correct message.*

PROOF. We prove that Byzantine nodes cannot manipulate validated messages.

Consider s is the last Byzantine node in a path P between node u and node w. We claim that if a message m with label l is delivered to w successfully through P, then s must appear in the flooding path of this message because node s is recorded in the flooding path by the next correct node after node s in P. This information will not be modified by other nodes in the graph because s is the last Byzantine node in P.

Byzantine nodes appear in at most f disjoint paths. Thus, at most f copies of m with label l can be modified by them. However, we need ($f + 1$) identical copies to validate message m with label

l. Therefore, there must be at least one copy of m is correct. This message is transmitted through a path without any Byzantine node.

There is at least one correct copy of m with label l accepted by node w when m with label l is validated by node w. Hence, message m is a correct message sent by node u if it is validated in the asynchronous purifying algorithm. □

Until now, we accomplished the randomized protocol that solves the Byzantine agreement problem with the presence of f Byzantine nodes in our model.

THEOREM 5. *If the communication graph G of the asynchronous distributed system with n nodes and $f < \frac{n}{3}$ Byzantine nodes have ($2f + 1$) vertex connectivity, then our algorithm solves the asynchronous Byzantine agreement problem, that all correct nodes decide on the same value with high probability.*

5 NECESSARY CONDITIONS

In the previous section, we presented a randomized algorithm that solves the Byzantine agreement problem with n nodes with the presence of f Byzantine nodes in our model where the connectivity of the communication graph is at least ($2f + 1$). This algorithm gives us a upper bound of the vertex connectivity requirement for the communication graph.

In this section, we prove that this bound is tight. There does not exist an algorithm that can solve the asynchronous Byzantine agreement problem with high probability when the vertex connectivity of G is less than ($2f + 1$).

THEOREM 6. *If the communication graph G of the asynchronous distributed system with n nodes and $f < \frac{n}{3}$ Byzantine nodes does not have ($2f + 1$) vertex connectivity, then it is impossible to find an algorithm that solves the asynchronous Byzantine agreement problem, i.e., all correct nodes decide on the same value.*

There are two different requirements in Theorem 6. We prove Theorem 6 with Lemma 11 and Lemma 12.

LEMMA 11 ([4]). *If there exists an algorithm that solves the Byzantine agreement problem in our model with n nodes with the presence of f Byzantine nodes, then $n \geq 3f + 1$.*

LEMMA 12. *If there exists an algorithm that solves the Byzantine agreement problem in our model with n nodes on the communication graph G in the presence of f Byzantine nodes, then G is ($2f + 1$)-connected.*

Lemma 11 was proved in [4]. It is impossible for randomized algorithms to solve the asynchronous Byzantine agreement problem if the number of Byzantine nodes f is greater than or equal to $\frac{n}{3}$.

We prove Lemma 12 by using the state machine based approach, which is similar to the technique in [4, 8, 9]. We discuss an example that it is impossible for any randomized algorithm to solve the Byzantine agreement problem with high probability. Correct nodes will decide on different values.

PROOF FOR LEMMA 12. Suppose for the sake contradiction, there exists an algorithm A that solves the asynchronous Byzantine agreement problem with high probability in our model with n nodes, tolerating at most $f < \frac{n}{3}$ Byzantine faulty nodes, communicating

on the communication graph $G = (V_G, E_G)$, where G is not $(2f+1)$-connected. A outlines a procedure A_u for each node $u \in V_G$ that describes state transitions of u.

We can find a vertex cut of G and the size of the vertex cut is less than $(2f + 1)$. Let C be the vertex cut of G and $|C| \leq 2f$. Other vertices $V_G \setminus C$ are partitioned into two non-empty vertex sets X, Y, such that X and Y are disconnected in $G \setminus C$.

Because $|C| \leq 2f$, we can partition C into two disjoint vertex sets R and T, i.e., $C = R \cup T$ and $0 < |R| \leq f$ and $0 < |T| \leq f$. The structure of network G is shown in Figure 4.
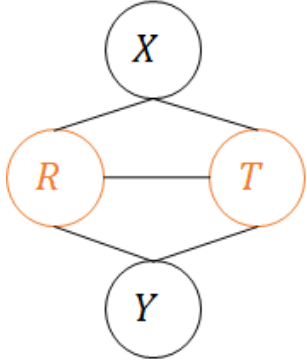

Figure 3: Network structure of G.

Let us consider three executions E_1, E_2, and E_3 on G. We run algorithm A in these three executions.

In the first execution E_1, nodes in T are Byzantine while other nodes in X, Y, R are correct with input 0.

In the second execution E_2, nodes in T are Byzantine, and other nodes in X, Y, R are correct with input 1.

In the third execution E_3, nodes in X, Y, T are correct, while nodes in R play Byzantine strategies. We input 0 to nodes of X and input 1 to nodes in Y and T.

It is easy for us to figure out the outputs of nodes in E_1 and E_2. Because of the validity of A, all correct nodes in E_1 decide on 0, no matter how Byzantine nodes act. For the same reason, all correct nodes in E_2 agree on 1.

However, it is not intuitive what is the output of nodes in the third execution E_3. We will explain that correct nodes in E_3 will output different values if Byzantine nodes in R play specific strategies. To understand the behavior of nodes in E_3, we introduce the fourth execution E_4 on another graph H.

We construct a network $H = (V_H, E_H)$ based on G. For each node $u \in V_G$, there are two copies of u, i.e., $u_0, u_1 \in V_H$. Thus, V_H can be partitioned into 8 vertex sets: $(X_0, Y_0, R_0, T_0, X_1, Y_1, R_1, T_1)$.

If $(u, v) \in E_G$ and u, v are in the same vertex set, then we copy (u, v) twice in E_H, such that $(u_0, v_0) \in E_H, (u_1, v_1) \in E_H$. If $(u, v) \notin E_G$, then there is no edge between (u_0, v_0) and (u_1, v_1).

If $(u, v) \in E_G$ and u, v are not in the same vertex set, we build connections according to the following rules.

- If $u \in R, v \in T$ and $(u, v) \in E_G$, then $(u_0, v_0) \in E_H, (u_1, v_1) \in E_H$.
- If $u \in X, v \in R$ and $(u, v) \in E_G$, then $(u_0, v_0) \in E_H, (u_1, v_1) \in E_H$.
- If $u \in Y, v \in R$ and $(u, v) \in E_G$, then $(u_0, v_0) \in E_H, (u_1, v_1) \in E_H$.
- If $u \in Y, v \in T$ and $(u, v) \in E_G$, then $(u_0, v_0) \in E_H, (u_1, v_1) \in E_H$.
- If $u \in X, v \in T$ and $(u, v) \in E_G$, then $(u_1, v_0) \in E_H, (u_0, v_1) \in E_H$.

We give the structure of H in Figure 5. Edges within vertex sets are not displayed, while edges between vertex sets are represented by a single edge.
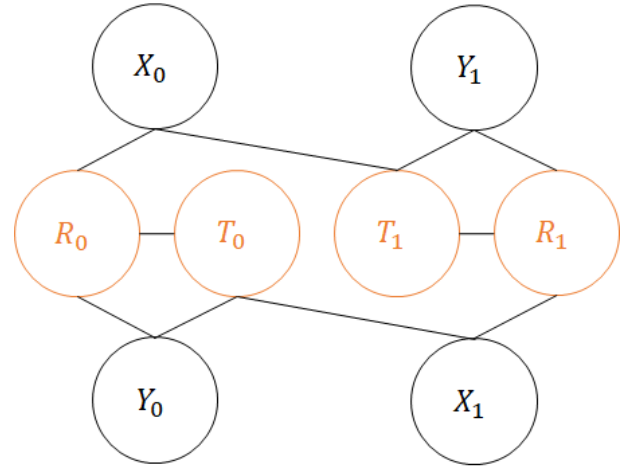

Figure 4: Network structure of H to run execution E_4.

E_4 is an execution on H as follows. Each node pair $u_0, u_1 \in V_H$ runs A_u, which is the same algorithm protocol as $u \in V_G$ runs. Nodes in X_0, Y_0, R_0, T_0 have initial input 0 and nodes in X_1, Y_1, R_1, T_1 have initial input 1. All nodes are correct.

To understand the output of nodes in E_3, we first discuss the output of nodes in E_4. We start from nodes in X_0, Y_0, R_0 in E_4. We claim that the behavior of X_0, Y_0, R_0 in E_4 is modelled by X, Y, R in E_1.

Let us consider strategies of Byzantine nodes in T in E_1. Each node $u \in T$ in E_1 can play a mixed strategy of the actions of $(u_0 \in T_0, u_1 \in T_1)$ in E_4. It considers itself as a combination of u_0 and u_1 in E_4. It reacts to nodes in X as the same as the reaction of u_1 to nodes in X_0 and reacts to nodes in Y and R as the same as the reaction of u_0 to nodes in Y_0, R_0.

As we discussed before, no matter how nodes in T behavior, the output of nodes in X, Y, R in E_1 is 0. Because nodes in X_0, Y_0, R_0 in E_4 are in the same environment, have the same input, and run the same algorithm as nodes in X, Y, R in E_1, they will also decide on 0 eventually. This simulation is shown in Figure 6.
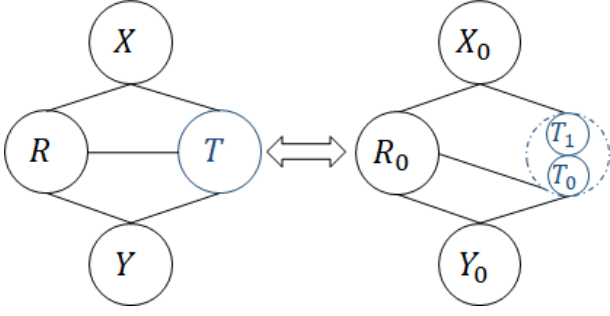
Figure 5: The behavior of nodes in E_1 and nodes in E_4. Green nodes are Byzantine nodes in E_1. T_v copies behavior of T_1 to X_0 and T_0 to Y_0, R_0. Nodes in X, Y, R decide on 0 eventually in E_1 while nodes in X_0, Y_0, R_0 decide on 0 eventually in E_4.

For the same reason, nodes in X_1, Y_1, R_1 decide on 1 eventually in E_4. We show the simulation of nodes in X_1, Y_1, R_1 in E_4 in Figure 7.
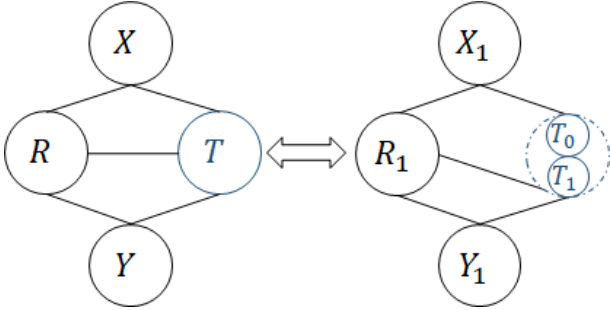

Figure 6: The behavior of nodes in E_2 and nodes in E_4. Green nodes are Byzantine nodes in E_2. T_v copies behavior of T_0 to X_1 and T_1 to Y_1, R_1. Nodes in X, Y, R deiced on 1 with high probability in E_2 while nodes in X_1, Y_1, R_1 deiced on 1 with high probability in E_4.

Each Byzantine node $u \in T$ in E_2 considers itself as a combination of u_0 and u_1 in E_4. It reacts to nodes in X as the same as the reaction of u_0 to nodes in X_1 and reacts to nodes in Y and R as the same as the reaction of u_1 to nodes in Y_1, R_1. No matter how nodes in T behave, the output of nodes in X, Y, R in E_2 is 1. Because nodes in X_1, Y_1, R_1 in E_4 are in the same environment, have the same input and run the same algorithm as nodes in X, Y, R in E_2, they will also decide on 1 eventually.

Based on previous analysis, we can design Byzantine strategies for nodes in R in E_3. Then nodes in X, Y, and T will decide on different values eventually. Each node $u \in R$ plays a mixed strategy of actions of $u_0 \in R_0$ and $u_1 \in R_1$ in E_4. It reacts to nodes in X as the same as the reaction of u_0 to nodes in X_0 and reacts to nodes in Y and T as the same as the reaction of u_1 to nodes in Y_1, T_1. This simulation is shown in Figure 8.
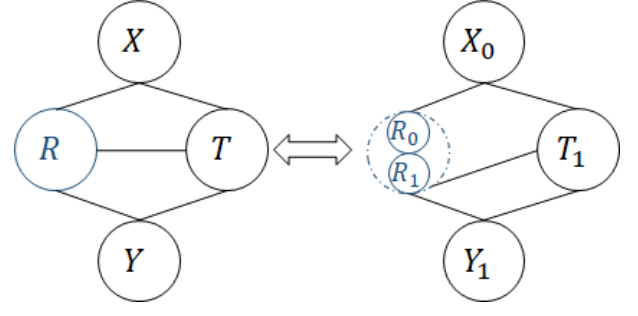

Figure 7: The behavior of nodes in E_3 and nodes in E_4. Green nodes in R are Byzantine nodes in E_3, which copy the behavior of R_0 and R_1 in E_4. correct nodes in X deiced on 0 and nodes in Y decide on 1 with high probability in E_3.

X_0, Y_1, T_1 in E_4 and X, Y, T in E_3 have the same input, the same algorithm procedure, and the same environment. Hence, nodes in X decide on 0, and nodes in Y decide on 1 eventually in E_3. This result indicates that A cannot achieve the Byzantine agreement in E_3, which contradicts the agreement property of A and our assumption.
□

In Lemma 11 and Lemma 12, we prove that $(2f + 1)$ vertex connectivity is a necessary condition for solving the asynchronous Byzantine agreement problem in an incomplete network with n nodes in the presence of $f < \frac{n}{3}$ Byzantine nodes.

6 CONCLUSION

In this work, we investigate the Byzantine agreement problem in the asynchronous distributed system with restricted communication.

Compare to previous work, we are the first to study the classical Byzantine agreement problem in a more realistic problem in world-scale distributed systems, i.e., blockchain systems.

We prove that following conditions are necessary and sufficient to achieve Byzantine agreement among n nodes with the presence of f Byzantine nodes: the communication graph has $(2f + 1)$-vertex connectivity and the number of Byzantine nodes f is less than $\frac{n}{3}$. We also present a randomized algorithm that solves the exact Byzantine agreement problem on incomplete graphs in asynchronous systems.

Beyond these contribution, we propose a three-layer framework that solves the asynchronous Byzantine agreement problem in incomplete networks. The bottom layer is a strong protocol which allows other algorithms [15] which solve the asynchronous Byzantine agreement problem to be implemented in incomplete networks.

One might hope that some classic synchronizers can help. However, classic synchronizers cannot deal with Byzantine nodes. There are algorithms [12] that solve the Byzantine synchronization, however, only in complete graphs, with additional assumptions.

To build a byzantine-tolerant synchronizer in an incomplete network seems to be an interesting problem, and we believe that our paper could be a stepping stone to understanding that problem. Though we cannot claim that we directly solve the problem, because the Byzantine clock synchronization problem can be considered as a Byzantine agreement problem with specified conditions.

REFERENCES

[1] Ittai Abraham, Yonatan Amit, and Danny Dolev. 2004. Optimal resilience asynchronous approximate agreement. In *International Conference On Principles Of Distributed Systems*. Springer, 229–239.

[2] Michael Ben-Or. 1983. Another advantage of free choice (Extended Abstract) Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*. 27–30.

[3] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.

[4] Gabriel Bracha and Sam Toueg. 1983. Resilient consensus protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*. 12–26.

[5] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. 2011. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media.

[6] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[7] Danny Dolev. 1982. The Byzantine generals strike again. *J. Algorithms* 3, 1 (1982), 14–30.

[8] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. 1986. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)* 33, 3 (1986), 499–516.

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

[10] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[11] Muhammad Samir Khan, Syed Shalan Naqvi, and Nitin H Vaidya. 2019. Exact byzantine consensus on undirected graphs under local broadcast model. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 327–336.

[12] Leslie Lamport and P Michael Melliar-Smith. 1985. Synchronizing clocks in the presence of faults. *Journal of the ACM (JACM)* 32, 1 (1985), 52–78.

[13] Leslie Lamport, Robert Shostak, and Marshall Pease. 2019. The Byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*. 203–226.

[14] Karl Menger. 1927. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae* 10, 1 (1927), 96–115.

[15] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. 2014. Signature-free asynchronous Byzantine consensus with t< n/3 and O (n2) messages. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*. 2–9.

[16] Achour Mostefaoui, Michel Raynal, and Frédéric Tronel. 2000. From binary consensus to multivalued consensus in asynchronous message-passing systems. *Inform. Process. Lett.* 73, 5-6 (2000), 207–212.

[17] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system*. Technical Report. Manubot.

[18] Muhammad Samir Khan and Nitin Vaidya. 2019. Asynchronous Byzantine Consensus on Undirected Graphs under Local Broadcast Model. *arXiv preprint arXiv:1909.02865* (2019).

[19] Russell Turpin and Brian A Coan. 1984. Extending binary Byzantine agreement to multivalued Byzantine agreement. *Inform. Process. Lett.* 18, 2 (1984), 73–76.