Byzantine Connectivity Testing in the Congested Clique

John Augustine
Department of Computer Science and Engineering,
Indian Institute of Technology Madras, Chennai, India

Anisur Rahaman Molla
Indian Statistical Institute, Kolkata, India

Gopal Pandurangan
Department of Computer Science, University of Houston, TX, USA

Yadu Vasudev
Department of Computer Science and Engineering,
Indian Institute of Technology Madras, Chennai, India

Abstract

We initiate the study of distributed graph algorithms under the presence of Byzantine nodes. We consider the fundamental problem of testing the connectivity of a graph in the congested clique model in a Byzantine setting. We are given a n-vertex (arbitrary) graph G embedded in a n-node congested clique where an arbitrary subset of B nodes of the clique of size up to $(1/3 - \varepsilon)n$ (for any arbitrary small constant $\varepsilon > 0$) can be Byzantine. We consider the *full information* model where Byzantine nodes can behave arbitrarily, collude with each other, and have unlimited computational power and full knowledge of the states and actions of the honest nodes, including random choices made up to the current round.

Our main result is an efficient randomized distributed algorithm that is able to correctly distinguish between two contrasting cases: (1) the graph $G \setminus B$ (i.e., the graph induced by the removal of the vertices assigned to the Byzantine nodes in the clique) is connected or (2) the graph G is far from connected, i.e., it has at least $2|B| + 1$ connected components. Our algorithm runs in $O(\text{polylog}\, n)$ rounds in the congested clique model and guarantees that all honest nodes will decide on the correct case with high probability. Since Byzantine nodes can lie about the vertices assigned to them, we show that this is essentially the best possible that can be done by any algorithm. Our result can be viewed also in the spirit of property testing, where our algorithm is able to distinguish between two contrasting cases while giving no guarantees if the graph falls in the grey area (i.e., neither of the cases occur).

Our work is a step towards robust and secure distributed graph computation that can output meaningful results even in the presence of a large number of faulty or malicious nodes.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Discrete mathematics

Keywords and phrases Byzantine protocols, distributed graph algorithms, congested clique, graph connectivity, fault-tolerant computation, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.7

Funding *John Augustine*: Supported in part by Extra-Mural Research Grant (file number EMR/2016/003016), MATRICS grant (file number MTR/2018/001198), and VAJRA faculty program funded by SERB, Government of India; also supported by the potential Centre of Excellence in Cryptography Cybersecurity and Distributed Trust (CCD) under the IIT Madras Institute of Eminence scheme.
Anisur Rahaman Molla: Research supported in part by DST Inspire Faculty research grant DST/INSPIRE/04/2015/002801 and ISI DCSW/TAC Project (file number E5412).
Gopal Pandurangan: This work was supported in part by NSF grants CCF-1717075, CCF-1540512, IIS-1633720, BSF grant 2016419, and by the VAJRA faculty program of the Government of India.

1 Introduction

Computation in the presence of faulty or malicious (also called *Byzantine* nodes) is a central issue in distributed computing. Indeed, since the introduction of the Byzantine agreement and broadcast problems in the early 1980s by Lamport, Pease, and Shostak [30, 24], Byzantine protocols have been studied extensively for the last four decades (see e.g, [10, 4, 21, 20, 2] and the references therein). These protocols have focused mainly on addressing fundamental, but "primitive" computational tasks such as agreement, leader election, broadcast etc in complete networks. Some of these protocols work in the CONGEST model as well (for e.g., the protocols of [21, 2]) where messages are of small size, i.e., polylog n size in networks of size n and still guarantee fast algorithms, running in polylog n rounds. Hence these Byzantine protocols solve these fundamental tasks of agreement and leader election in complete networks in the CONGEST model (i.e., using only small-sized messages).

In this paper we take a step towards addressing more complex computational tasks under presence of a large number of Byzantine nodes. In this direction, we initiate the study of distributed graph algorithms under the presence of Byzantine nodes.

We focus on the fundamental problem of testing the connectivity of a graph in the *congested clique* model in a *full information* Byzantine setting. Graph connectivity and related problems such as minimum spanning tree (MST) have been studied extensively in the congested clique model in the last decade or so (see e.g., [27, 17, 13, 18]). In this model, we have a network that is a clique on n nodes and an (arbitrary) input graph G having n vertices that is embedded in the clique (with each vertex of G mapped to a node of the clique). Throughout we use "nodes" to denote the processors of the congested clique and "vertices" to denote the vertices of the input graph G. Connectivity (as well as MST) of the input graph G can be solved very efficiently in the congested clique, i.e., in $O(\log n)$ rounds [31]. In fact, after a long line of research, it is now known that these problems can be solved even faster, i.e., in $O(1)$ rounds [18]. All known results in the congested clique (or in the closely related k-machine model [22, 23, 14, 28, 29] or in the Massively Parallel Computing (MPC) model [3, 25, 19, 1] used in distributed large-scale graph computations) assume that all processors are honest and faithfully participate in the computation. However, in many applications, e.g., in distributed big data computations by a large network of processors, some of the processors can be faulty or even malicious.[1] Motivated by such scenarios, we assume that an arbitrary subset of B nodes of the clique of size up to $(\frac{1}{3} - \varepsilon)n$ (for any arbitrary small constant $\varepsilon > 0$) can be *Byzantine*. We assume the *full information* model where Byzantine nodes can behave arbitrarily, collude with each other, and have unlimited computational power and full knowledge of the states and actions of the honest nodes, including random choices made up to the current round. Our goal is to study how meaningful graph computations can be accomplished efficiently (say, running in a small number of rounds) even in the presence of a large number of Byzantine nodes.

It is not a priori clear how to do meaningful graph computation under a large number of Byzantine nodes. We note that Byzantine nodes can arbitrarily deviate from the correct protocol. In particular, they can lie about the edges of G incident on themselves. For example, a Byzantine node can say that an edge incident on itself is not present to some honest nodes, while saying the contrary to some other honest nodes. They can also create

[1] The situation in the MPC or the k-machine model is harder compared to the congested clique model since the number of processors (some of which can be faulty) is much smaller than the number of nodes and thus each processor is responsible for many nodes in the input graph. This setting is left for future work, while the congested clique setting addressed here is a first step in this direction.

fake edges that are not originally present in G and selectively lie about them to honest nodes. Despite such malicious behavior we show that one can design efficient protocols such that all honest nodes end up computing non-trivial meaningful results.

Our main contribution is an efficient randomized Byzantine distributed protocol that is able to correctly distinguish between two contrasting cases: (1) the graph $G \setminus B$ (i.e., the input graph induced by the removal of the vertices assigned to the Byzantine nodes in the clique) is connected or (2) the graph G is far from connected, i.e., it has at least $2|B| + 1$ connected components.[2]

Since Byzantine nodes can lie about the vertices assigned to them, we show that the above gap between the two cases is essentially the best possible that can be done by any algorithm. Our algorithm is efficient in the sense that it takes only $O(\text{polylog}\, n)$ rounds in the n-node congested clique and guarantees that *all* honest nodes will decide on the correct case with high probability. (The exact power of the $\text{polylog}\, n$ depends on Byzantine agreement and committee election protocols used in prior works that also takes $\text{polylog}\, n$ rounds (cf. Section 2.2).) Our focus in this paper, as in prior works on agreement and committee election [21, 8]), is to show that the more challenging problem of Byzantine connectivity testing can also be accomplished efficiently, i.e., in $\text{polylog}\, n$ rounds.

Our result can be viewed also in the spirit of *property testing*. In the property testing model ([16, 32]) one has query access to an input, say a graph. The goal is to test whether the input has a property, or is far (in an appropriately defined way) from the property by querying a small part of the input. Property testing algorithms in this setting has been studied for a variety of problems (see [15] for a detailed exposition). Property testing algorithms in the CONGEST model was introduced by Censor-Hillel et al. ([6]), and algorithms for properties like connectivity, cycle-freeness, bipartiteness, planarity have been studied ([9, 11, 26]). Our current work extends this to testing connectivity in the presence of Byzantine nodes. Our algorithm checks if the input graph G with $B \subseteq V$ Byzantine nodes is far from connected or whether $G \setminus B$ is connected; we will precisely define the notion of being far from connected momentarily.

1.1 Model and Problem Statement

We assume the well-studied synchronous congested clique model consisting of n nodes (representing processors or computing devices) with each node identified by a unique ID from $\{1, 2, \ldots, n\} = [n]$. (The choice of $[n]$ as the space of IDs is without loss of generality with respect to the usual assumption that IDs are from a $\text{poly}(n)$ space because nodes know each others' IDs and can use their ordinal numbers in the sorted ordering of IDs instead of actual IDs.) Communication happens over synchronized rounds, where in each round a node can communicate with any of the other $n - 1$ nodes by sending a message of $O(\log n)$ bits (i.e., the CONGEST model). We are given an arbitrary (input) graph G which is embedded in the clique and is known only locally to the nodes in the network. Every vertex in G is mapped to a node (processor) in the clique and each clique node has knowledge of the edges incident to the vertex of G mapped to it. We assume that an arbitrary subset of B nodes of the clique of size up to $(\frac{1}{3} - \varepsilon)n$ (for any arbitrary small constant $\varepsilon > 0$) can be *Byzantine*. We assume that Byzantine nodes can behave arbitrarily, collude with each other, and have

[2] It might seem a bit unnatural that in the problem definition the yes and no instances consider two different graphs: G and $G \setminus B$. However, we cannot say anything about G itself – whether it is connected or not – (as is usually the case in a non-Byzantine setting), since even the presence of one Byzantine node can lead to both possibilities.

unlimited computational power and full knowledge of the states and actions of the honest nodes, including random choices made up to the current round. This is the so-called *full information* model. We note that the Byzantine nodes in the clique are chosen after the assignment of the graph vertices to the clique.

Our goal is to test properties of the input graph G despite the presence of Byzantine nodes. The complexity measure of interest is the number of rounds to decide if G has the property. We would like to design efficient algorithms, i.e., running in a small number of rounds. In this paper, we describe an algorithm to test connectivity of a graph G in the presence of Byzantine nodes. We would like *all* honest nodes to output a common "meaningful" decision which we define next.

Since Byzantine nodes can lie about the edges incident on them, it is not possible to test connectivity in the usual sense. For example, assume that G has a cut edge and both the endpoints are assigned to Byzantine nodes. Then the Byzantine node can lie about the presence of this edge to other (honest) nodes. Hence honest nodes may conclude that the graph is not connected.[3] More generally, there is no way to verify whether the subgraph of G induced on the set of Byzantine nodes is indeed connected or not. This example, motivates that we have to relax the notion of testing connectivity as follows.

The notion of connectivity that we would like to solve is the following. Let b, be the (upper bound) number of Byzantine nodes.[4] Our goal is to test whether $G \setminus B$ is connected or G is "far from connected." We say that G is *f-far from connected*, if one must add at least f edges to G in order to make it connected (in other words, there are at least $f+1$ components in G). All honest (good) nodes must report correctly when $G \setminus B$ is connected (output TRUE) or G is f-far from connected where $f = 2b$ (output FALSE). If neither of these hold, then in this case, all honest nodes can output either TRUE or FALSE, but they should output a common decision value. We will show in Theorem 3 that this gap is essentially the best possible for any algorithm to distinguish.

While the above gap is the best possible, it helps one to distinguish between two reasonably contrasting cases. If the input graph G is well connected, say for example, it is $b+1$-connected, then the algorithm will correctly output that $G \setminus B$ is connected, since no matter which vertices are assigned to Byzantine nodes, deletion of those vertices will still leave the graph connected. On the other hand, if G far from connected, i.e., it has lot of connected components (at least $2b+1$ of them), then again, no matter which vertices are assigned to the Byzantine nodes, then the algorithm will correctly identify that G is far from connected. If neither of these scenarios occur, then the algorithm will output an arbitrary value (but consistent across all honest nodes).

1.2 Our Results

Our main result is the following theorem.

▶ **Theorem 1.** *Given a graph G embedded in an n-node congested clique, out of which an (arbitrary) subset of B nodes ($|B| < (1/3 - \epsilon)n$), for any arbitrary small constant $\varepsilon > 0$, are Byzantine, we present a randomized Byzantine protocol, that with high probability[5], runs in $O(\text{polylog}\, n)$ number of rounds and outputs:*

[3] Note that the two Byzantine nodes assigned to the endpoints of the cut edge may lie to some honest nodes and not to others to prevent the honest nodes from reaching a common decision which is required; this is also an issue that one has to contend with.

[4] We assume that honest nodes have knowledge of this upper bound on the number of Byzantine nodes. Note that, in general, this is the best possible, since some Byzantine nodes can act as good nodes throughout the protocol.

[5] Throughout, "with high probability" means "with probability at least $1 - 1/n^c$" for some constant $c > 0$.

- *TRUE if $G \setminus B$ is connected;*
- *FALSE if G is $2|B|$-far from connected;*
- *TRUE or FALSE if neither of the above hold.*

All honest nodes output the same answer with high probability.

We also present a simple and deterministic solution. The result is stated in the following theorem. We defer a proof of this theorem to an extended version of this paper.

▶ **Theorem 2.** *Given a graph G embedded in an n-node congested clique, out of which an (arbitrary) subset of B nodes are Byzantine, where $|B| < n/3$. There is a deterministic algorithm that tests the connectivity of G in $O(n)$ rounds and outputs:*

- *TRUE if $G \setminus B$ is connected;*
- *FALSE if G is $2|B|$-far from connected;*
- *TRUE or FALSE if neither of the above hold.*

We note that the usual bound of $|B| < n/3$ Byzantine nodes in the full information model [30] for agreement holds here as well.

The gap in between the two extreme cases, i.e., whether $G \setminus B$ is connected or if it is $2|B|$-far from connected is the best possible that one can achieve in the presence of B Byzantine nodes. We defer the proof to the appendix of this paper.

▶ **Theorem 3.** *There is no algorithm that is guaranteed to distinguish whether a graph G embedded in an n-node congested clique containing a set B of b Byzantine nodes is f-far from being connected, or whether $G \setminus B$ is connected with probability at least $1/2 + \epsilon$ (for any fixed $\epsilon > 0$), unless $f \geq 2b$.*

1.3 Technical Challenges and Overview

The main difficulty in designing a protocol as claimed in Theorem 1 is correctly distinguishing the two cases or conclude it is in the gray area. If there are no Byzantine nodes, then one can simply run a distributed connectivity algorithm, in particular, a distributed minimum spanning tree (MST) algorithm (assuming all weights are 1) such as the Gallagher-Humblet-Spira (GHS) algorithm [12] – which is essentially the distributed Boruvka algorithm – on the congested clique (see e.g., [31]). This algorithm can be easily implemented to run in $O(\log n)$ rounds (deterministically) in the congested clique. However, in the presence of Byzantine nodes, this algorithm does not work.

There are two main challenges. First is to ensure that the MST algorithm correctly operates in a consistent manner despite the Byzantine nodes lying (selectively as well) about the presence or absence of edges incident to the vertices assigned to them. Second is to ensure that the algorithm operates fast, i.e., in $O(\text{polylog}\, n)$ rounds in the congested clique.

To illustrate the difficulties, as a warm up, we outline an algorithm (the detailed algorithm and its proof appears in the full version of the paper) that correctly solves the problem (i.e., it outputs correctly TRUE or FALSE depending on the cases), but takes polynomial, i.e., $O(n)$ rounds. The algorithm is deterministic. The high-level idea behind this algorithm is as follows. Each node acts as a leader and verifies all the edges of the graph. Byzantine nodes can try to foil the above algorithm by suggesting fake edges, i.e., edges that don't exist. This can be done selectively by sending different information to different honest nodes. Byzantine nodes can also reject edges that are suggested by good nodes. To overcome this, an honest node needs to verify if an edge is valid by querying both its endpoints; if both endpoints validate the edge then it is accepted; otherwise it is rejected. We show that adding

a validated or qualified edge is fine for the correctness of the algorithm; but this is not true for non-validated edges and hence they should not be added. At the end of the computation, an honest node will have all the validated edges. We show that an honest node can decide correctly by checking whether the *largest connected component* has at least $n - b$ nodes.

The above gives a fairly straightforward algorithm, though different honest nodes might decide on different answers (because of selective lying by Byzantine nodes). However, we show that if the input graph G falls in one of the cases, then all honest nodes will decide on the same answer. However, if the input graph falls in the gray area, then different honest nodes can end up with different answers. However, one can resolve this, by performing an efficient Byzantine agreement protocol. To keep the solution deterministic, we use a deterministic agreement protocol by Dolev et al. [8].

A main drawback of the above approach is that, validating all the edges can cause lot of congestion. For example, it can happen that $\Theta(n)$ edges have a common endpoint and this information has to be conveyed to every honest node which verifies it. This takes at least $\Theta(n)$ rounds.

Our main contribution is a significantly faster algorithm that runs in $O(\text{polylog}\, n)$ rounds that is able to avoid congestion and still correctly output the desired answer. At the beginning of the algorithm, we elect a *leader committee* of $O(\log n)$ leaders. The key technical part of the algorithm is for each one of the leaders to decide YES/NO. Then the $O(\log n)$ leaders will do Byzantine agreement among themselves to decide the final output (that all honest nodes can sample). Each leader implements the distributed version of Boruvka's algorithm ([12]) on the graph, but instead of each node running the algorithm, we create committees of nodes[6], and delegate the algorithm to these committees. The crucial technical ingredient is a way to construct $O(\log n)$-sized committees for each of the nodes such that the fraction of committee members that are honest is proportional to the actual fraction of honest nodes in the network. The leader samples $O(\log n)$ many hash functions which are broadcast to every node in the network. The honest nodes can use these to compute the committee pertaining to each node in the network. This makes the committees common knowledge among the nodes in the network, and prevents any tampering by Byzantine nodes. Furthermore, our construction ensures that each of the committees have a consistent view of the edges in the graph. We will then run the algorithm constructing the spanning tree via these committees.

2 Preliminaries

2.1 Tail inequalities and hash functions with limited independence

We make use of hash functions with limited independence to save messages (and hence avoid congestion). These hash functions use c-wise independence and hence we use the following tail inequalities and properties of such hash functions.
The following tail inequalities are from [33].

▶ **Lemma 4.** *Let $c \geq 4$ be an even integer. Suppose $Z_1, Z_2, \ldots, Z_t$ are c-wise independent random variables taking values in $[0, 1]$. Let $Z = \sum_{i=1}^{t} Z_i$ and $\mu = \mathbb{E}[Z]$, and let $\lambda > 0$. Then,*

$$Pr[|Z - \mu| \geq \lambda] \leq 2\left(\frac{ct}{\lambda^2}\right)^{c/2}.$$

[6] The idea of communicating using committees has been used before, see e.g., [5, 20]. Note that these "node" committees are different from the leader committee, elected once at the beginning.

▶ **Lemma 5.** *Suppose that X is the summation of n 0-1 random variables $X_1, X_2, \ldots, X_n$, each with mean p. Let μ satisfy $\mu \geq \mathbb{E}[X] = np$. If the X_is are $\mu\delta$-wise independent, then for every $\delta \geq 1$*

$$\mathbb{P}[X \geq (1+\delta)\mu] \leq exp(-\delta\mu/3\}).$$

The following is Definition 7 in [7].

▶ **Definition 6.** *For N, L, $c \in \mathbb{N}$, such that $c \leq N$, a family of functions $\mathcal{H} = \{h : [N] \to [L]\}$ is c-wise independent if for all distinct $x_1, x_2, \ldots, x_c \in [N]$, the random variables $h(x_1), h(x_2), \ldots, h(x_c)$ are independent and uniformly distributed in $[L]$ when h is chosen uniformly at random from H.*

The following lemma appears as Corollary 3.34 in [34].

▶ **Lemma 7.** *For every a, b, c, there is a family of c-wise independent hash functions $\mathcal{H} = \{h : \{0,1\}^a \to \{0,1\}^b\}$ such that choosing a random function from $\mathcal{H}$ takes $c \cdot \max\{a, b\}$ random bits, and evaluating a function from $\mathcal{H}$ takes $poly(a, b, c)$ computation.*

2.2 Byzantine agreement and committee election

As a subroutine in our protocols, we utilize the following results from King et al. [21] that gives efficient protocols (running in $O(\text{polylog}\, n)$ rounds) for Byzantine agreement and committee election in a n-node complete network (i.e., congested clique) that can tolerate up to $(1/3 - \epsilon)n$ Byzantine nodes in the full information model. We note that the protocol of King et al. has the property that every honest node sends and processes only $O(\text{polylog}\, n)$ number of bits throughout the course of the algorithm. This constrains them to achieving only almost-everywhere agreement where only $1 - o(1)$ fraction of the honest nodes agree. If we allow each node of the clique to send $O(\log n)$ bits per clique edge per round, then one can achieve everywhere agreement (i.e., all honest nodes agree) in a straightforward manner [21]. Similarly, in the committee election problem, all honest nodes can be informed of the identities of the committee nodes.

For the sake of completeness, we first define Byzantine agreement and Byzantine committee election.

In *Byzantine (everywhere) agreement*, starting with each node having an input value (0 or 1), the goal is for all honest nodes to agree on a common value, which should also be an input value of some honest node.

In *Byzantine committee election*, the goal is to elect a committee of size $\Theta(\log n)$ such that with high probability, the committee consists of at least $\frac{2}{3}$ fraction of honest processors.

▶ **Theorem 8** (Byzantine committee election [21])**.** *Consider a n-node congested clique model where up to $(1/3 - \epsilon)n$ (for any arbitrary small constant $\varepsilon > 0$) of the nodes can be Byzantine in the full information model. Then there is a randomized protocol that elects a committee of size $\Theta(\log n)$ that, with high probability, contains at least 2/3-fraction of honest nodes in the committee. The protocol runs in $O(\text{polylog}\, n)$ rounds and all honest nodes in the network know the identities of the committee members.*

▶ **Theorem 9** (Randomized Byzantine everywhere agreement [21])**.** *Consider a n-node congested clique model where up to $(1/3 - \epsilon)n$ (for any arbitrary small constant $\varepsilon > 0$) of the nodes can be Byzantine in the full information model. There is a randomized protocol that, with high probability, solves Byzantine agreement such that all honest nodes in the network agree on a common value (which will be the input value of a honest node) and runs in $O(\text{polylog}\, n)$ rounds.*

We also use a *deterministic* protocol that solves Byzantine everywhere agreement that takes linear (in the size of the network) number of rounds. We use this protocol only on a $O(\log n)$-size network (i.e., a committee of size $O(\log n)$) and hence the time taken is $O(\log n)$ rounds.

▶ **Theorem 10** (Deterministic Byzantine Everywhere Agreement [8]). *Consider a k-node congested clique model where fewer than $k/3$ of the nodes can be Byzantine in the full information model. There is a protocol that deterministically solves Byzantine agreement such that all honest nodes in the network agree on a common value (which will be the input value of a honest node) and runs in $O(k)$ number of rounds.*

3 An $O(\text{polylog}\, n)$-round Algorithm

In this section we give a $O(\text{polylog}\, n)$-round algorithm for testing connectivity by building committees for each vertex in the graph such that the communication between nodes is delegated to their respective committees. We ensure that the committee members are chosen randomly so that all committees, with high probability, comprise more than 2/3-fraction of good nodes. Thus, even bad nodes are represented by good committees and this significantly limits the power of the Byzantine nodes.

We first build a leader committee L of $O(\log n)$ size so that more than a 2/3-fraction of the nodes in the committee are good; see Algorithm 1. This can be done in $O(\text{polylog}\, n)$ rounds using the protocol of [21] (cf. Theorem 8). This protocol also ensures that all honest nodes know the identities of all the committee members. Now each $\ell \in L$ acts as a leader and orchestrates the execution of distributed Boruvka algorithm and decides on the output; thus, Boruvka's algorithm is invoked independently by each $\ell \in L$. This is the key technical part of the algorithm that we explain next. Once all nodes $\ell \in L$ arrive at their respective decisions, they can perform a deterministic Byzantine agreement protocol among the committee members using the protocol of [8] (cf. Theorem 10). We will run the protocol of [8] for each node in the leader committee acting as a *transmitter*.[7] This guarantees consistent agreement among the committee nodes of the value of this transmitter node. (The running time will be $O(\log^2 n)$ rounds since one run of the transmitter protocol takes linear number of rounds in the size of the committee which is $O(\log n)$.) Since the majority of the nodes in the leader committee are honest, this also ensures that the majority answer is the correct answer agreed by all honest nodes in the committee. All other honest nodes can then query all the committee nodes and then take their majority output as their own output. The formal steps are outlined in Algorithm 1.

We now briefly comment on how each $\ell \in L$ orchestrates the execution of Boruvka's algorithm. We show that it is possible for each ℓ to construct committees of size $O(\log n)$ for each vertex with the property that if at most a β fraction of the nodes in the network are Byzantine for a fixed constant $\beta < 1/3$, then, with high probability, in every committee at most $\beta + \epsilon < 1/3$ fraction of nodes are Byzantine (for a small constant $\epsilon > 0$). Moreover, each committee $\mathsf{Comm}(v)$ learns the edges incident at the node v that it represents and can interact with committees representing v's neighbors. In other words, the committees (in a collective sense) learn a consistent view of the graph. Finally, the nodes are balanced across the committees in the sense that no node needs to be a member of more than $O(\log^2 n)$ committees. Theorem 15 proves all these properties formally. We then show how to construct the spanning tree using these committees and thus test connectivity.

[7] This is to ensure majority consensus, i.e., the agreed value is also the majority value among the inputs, if such a majority exists.

Algorithm 1 The main stem.

1: Use the protocol in [21] to elect a leader committee L of size $\Theta(\log n)$. /* (cf. Theorem 8 for relevant properties.) */
2: **for** each $\ell \in L$ in parallel **do**
3: Invoke Algorithm 8 with leader ℓ.
4: /* ℓ learns either *accept* or *reject* as its outcome */
5: The nodes in L perform Byzantine agreement on their outcomes using [8] (as outlined above) to reach their *final outcomes*.
6: Each $\ell \in L$ sends their final outcome to all nodes in $[n] \setminus L$.
7: All nodes in $[n] \setminus L$ set their final outcomes to the value sent by a majority in L.

3.1 Committees Takeover

We now consider the case where a single leader ℓ is able to produce a committee structure that maps every node v in the network to its representative $\Theta(\log n)$-sized committee $\mathsf{Comm}(v)$ comprising sufficiently many good nodes. Moreover, this committee structure must be common knowledge in the sense that every node u should be able to compute $\mathsf{Comm}(v)$ for all nodes v. Each $\mathsf{Comm}(v)$ can then perform computation and communication on behalf of the node v that it represents. However, each $\mathsf{Comm}(v)$ must first learn the required information (such as the neighborhood of v). If v is Byzantine, the information $\mathsf{Comm}(v)$ learns may be incorrect and inconsistent, but the committees should ensure consistency and also ensure correctness as far as possible.

We assume the following throughout this section. Let $\beta < 1/3$ and $\varepsilon < 1/3 - \beta$ be fixed constants; thus, $\beta + \varepsilon$ is fixed and strictly below $1/3$. We assume that, out of a total of $|V| = n$ nodes, the number of Byzantine nodes is at most βn. Let η be a sufficiently large constant that depends on β and ε, and let $k = \lceil \eta \log n \rceil$. Recall that the set of nodes in the network are $[n]$. When we refer to vertex $v \in [n]$, we are referring to the vertex in G that is assigned to v. Similarly, for $u \in [n]$ and $v \in [n]$, we use $(u, v) \in E(G)$ (resp., $(u, v) \notin E(G)$) to denote that the vertices assigned to u and v form an edge (resp., do not form an edge) in G. Finally, we use $N(v)$ to denote the set $\{u \in [n] \setminus \{v\} \mid (v, u) \in E(G)\}$.

The leader ℓ uses k-wise independent hash functions (cf. Definition 6) in order to build the committee structure. From Lemma 7, we know that there is a k-wise independent family of hash functions $\mathcal{H}$ with each $h \in \mathcal{H}$ mapping $[n] \to [n]$. Moreover, picking a random h only requires $k + 2\lceil \log n \rceil \in O(\log n)$ bits. Thus, ℓ generates the required $\Theta(\log^2 n)$ random bits that can be used to pick k random hash functions $h_1, h_2, \ldots, h_k$. The leader then broadcasts those $O(\log^2 n)$ bits to all nodes in the network, thereby making the k hash functions common knowledge amongst all the nodes. The committee for each node $v \in [n]$ comprises nodes in $\{h_i(v) | 1 \leq i \leq k\}$, thereby making the entire committee structure common knowledge. Thus, we can summarize the properties of the committee structure as follows.

▶ **Lemma 11.** *The committee structure created by a good leader node ℓ comprises commonly known committees $\mathsf{Comm}(v)$ for all $v \in V$ such that $|\mathsf{Comm}(v)| \leq k \in O(\log n)$. Moreover, the randomness in the choice of committee members ensures WHP that:*

1. *the proportion of Byzantine nodes within each committee is at most $\beta + \varepsilon + o(1) < 1/3$, and*
2. *for every $c \in V$, $C(c) \triangleq \{v | c \in \mathsf{Comm}(v)\}$, i.e., the set of nodes that c represents as a committee member, has a cardinality of at most $O(\log^2 n)$.*

Algorithm 2 COMMITTEES-LEARN-INCIDENT-EDGES.

Require: Assume a good leader ℓ is elected and known to all nodes. Let $k = \lceil \eta \log n \rceil$ for a sufficiently large but fixed η.

Ensure: For each $v \in V$, an associated committee $\mathsf{Comm}(v)$ is created. See Theorem 15 for a detailed listing of all other guarantees.

The leader ℓ generates $\Theta(\log n)$ bits drawn uniformly and independently at random and broadcasts them to all other nodes.

All nodes v use those $\Theta(\log n)$ bits to draw $h_1, h_2, \ldots, h_k$ from the k-wise independent family $\mathcal{H}$ of hash functions.
/* Thus, any node v can compute the committee $\mathsf{Comm}(u) = \{h_i(u) | 1 \leq i \leq k\}$ associated with any node u. */

In parallel $\forall v \in V$ **do**

    Node v sends the cardinality of its neighborhood $|N(v)|$ to every $c \in \mathsf{Comm}(v)$.

    In parallel $\forall c \in \mathsf{Comm}(v)$, and $\forall i \in \{1, 2, \ldots, |N(v)|\}$ **do**

        c elects a *routing committee* $\mathsf{RouteComm}(c, v, i)$ of cardinality $\Theta(\log n)$ WHP in the following manner. For every $u \in V \setminus \{c, v\}$, $\mathbb{P}[u \in \mathsf{RouteComm}(c, v, i)] = \Theta(\frac{\log n}{n})$, independently over all u and all i.

        c sends a request for ith neighbor of v to all members in $\mathsf{RouteComm}(c, v, i)$.

        In parallel each $r \in \mathsf{RouteComm}(c, v, i)$ **do**

            if $c \in \mathsf{Comm}(v)$ **then**

                r conveys the message from c to v.

            Let q be the number of request messages c conveyed to v through r.

            if $q \in O(\log n)$ **then**

                v responds to r's request with the ID of its ith neighbor.

            r relays the responses back to c.

        End parallel

        Let μ be the ID reported by most nodes in $\mathsf{RouteComm}(c, v, i)$.

        if a majority in $\mathsf{RouteComm}(c, v, i)$ reported μ **then**

            c records the ith neighbor as μ.

        else

            Reject the response.

                /* c can conclude that v is Byzantine. */

    End parallel

End parallel

In parallel $\forall$ pairs $u \in V$ and $v \in V$ **do**

    Ensure all good nodes in $\mathsf{Comm}(u)$ and $\mathsf{Comm}(v)$ have a consistent record of whether edge $(u, v) \in E(G)$. Note that if both u and v are good, then their record, in addition to being consistent, must be correct. This is achieved by calling Algorithm 6.

End parallel

In order to execute graph algorithms, we require one more crucial ingredient. For each vertex v, the members of $\mathsf{Comm}(v)$ must learn the list of neighbors of v and that procedure is specified in Algorithm 2. One may be tempted to think that this can be achieved simply by v sending its adjacency list to each $c \in \mathsf{Comm}(v)$, but this will take time proportional to v's degree, which can be linear in n leading to an algorithm that is linear in n. To sidestep this delay, we can try random routing to parallelize this process; node v communicates the

ID of each neighbor of v to each $c \in \mathsf{Comm}(v)$ through a randomly chosen intermediate node. However if the random intermediate node is Byzantine, the information may be misrepresented. To overcome this, we use randomly chosen *routing committees* as formally described in Algorithm 2. Some care must be exercised because routing committees can also contain Byzantine nodes and the information passing through them has to therefore be vetted.

▶ **Lemma 12.** *By the end of the first parallel loop in Algorithm 2 ending in line number 23, every $c \in \mathsf{Comm}(v)$ for $v \in [n]$ has learned a list of neighbors of v with the guarantee that if both c and v are good, the list matches $N(v)$ correctly. Moreover, the time taken for that first parallel loop to complete is $O(\text{polylog}(n))$ WHP.*

At the end of the first parallel loop (line number 23), the nodes in the committees pertaining to two nodes u and v may not reach the same conclusion about whether $(u, v) \in E(G)$ or not. To ensure consistency and correctness (to the extent possible), we perform some agreement and consistency tie-breaks in the second parallel loop (post line number 23). To implement this second loop, we need Algorithm 6, which in turn utilizes some key primitives. The main idea is to first ensure that for each u, the committee $\mathsf{Comm}(u)$ first reaches an agreement on whether there is an edge from u to each v. Each committee therefore has to perform $n - 1$ agreements, thereby requiring a total of $n(n - 1)$ agreements. These large number of agreements are performed in parallel in $O(\text{polylog}(n))$ time (WHP) using Algorithm 5.

We begin with some simpler primitives that will then be used by Algorithm 5 and Algorithm 6. Our first primitive is Algorithm 3 where each the committee members of a node u can communicate messages pertaining to all other nodes v in $O(\text{polylog}(n))$ rounds.

Algorithm 3 Communication between committee members across committees. This algorithm is described from the perspective of a node c.

Require: For every $u \in V$, $c \in \mathsf{Comm}(u)$, and $v \in V$, the node c has a message $m(c, u, v)$. /* There are $O(n^2 \log n)$ such messages WHP. */

Ensure: Every $c' \in \mathsf{Comm}(v)$ must learn $m(c, u, v)$ as long as c is good.

1: **for** each $u \in C(c)$ and each $c' \in \mathsf{Comm}(v)$ **do** /* Recall that for each $c \in V$, $|C(c)| \in O(\log^2 n)$ WHP by Lemma 11. */
2: Node c sends $m(c, u, v)$ to c'.
3: Node c receives $m(c', v, u)$ sent by c'.

▶ **Lemma 13.** *Algorithm 3 terminates in $\Theta(\log^4 n)$ rounds WHP. Moreover, for every good c, all its messages are correctly conveyed (i.e., all messages are passed to the intended recipient).*

The second primitive is Algorithm 4 wherein for each u, the members of $\mathsf{Comm}(u)$ broadcast messages to each other. Specifically, each $c \in \mathsf{Comm}(u)$ has a message pertaining to the pair u and v. Thus, c is in possession of $O(n)$ messages under the role of $c \in \mathsf{Comm}(v)$; it may have other such messages as a member of other committees. All those $O(n)$ messages must reach all other $c'' \in \mathsf{Comm}(u)$. Due to the number of messages, c cannot directly send them to c''. Instead, c sends each message pertaining to the pair u and v to the members of $\mathsf{Comm}(v)$ using Algorithm 3 and then the members of $\mathsf{Comm}(v)$ echo them back to the members in $\mathsf{Comm}(u)$. When c and c'' are good, clearly, c'' will hear a majority of consistent messages correctly from c'' because the committees that are used to echo the messages are all dominated by good nodes. Furthermore, Algorithm 4 only requires $O(\log n)$ calls to Algorithm 3, thereby allowing us to conclude the following.

▶ **Lemma 14.** *For every pair u and v, every good node $c \in \mathsf{Comm}(u)$ will be able to broadcast its messages of the form $m(c, u, v)$ (see Algorithm 4) to all nodes in $\mathsf{Comm}(u)$ using Algorithm 4. Moreover, its complexity is $O(\log^5 n)$ rounds WHP.*

Algorithm 4 Broadcasting within each committee.

Require: For every $u \in V$, $c \in \mathsf{Comm}(u)$, and $v \in V$, the node c has a message $m(c, u, v)$.
/* As before, there are $O(n^2 \log n)$ such messages WHP. */

Ensure: For all u, every $c' \in \mathsf{Comm}(u)$ must learn $m(c, u, v)$ $(\forall c \in \mathsf{Comm}(u))$ correctly as long as c is good. /* Note that each c will possess $O(n \log n)$ messages of the form $m(c, \cdot, \cdot)$ and it must learn $O(n \log n)$ messages. */

/* To describe this algorithm, we focus on a pair u and v and a $c \in \mathsf{Comm}(u)$ and trace each copy of $m(c, u, v)$ as it makes its way to all other $c'' \in \mathsf{Comm}(u)$. This process must be executed in parallel for every u and v and every $c \in \mathsf{Comm}(u)$. For this reason, note that $m(c, u, v)$ cannot be directly sent to all $c'' \in \mathsf{Comm}(u)$ because there are a linear in n number of messages of the form $m(c, u, \cdot)$ that must reach c''. */

The message $m(c, u, v)$ reaches each $c' \in \mathsf{Comm}(v)$. This requires one invocation of Algorithm 3.

Each $c' \in \mathsf{Comm}(v)$ now sends $m(c, u, v)$ to each $c'' \in \mathsf{Comm}(u)$. This requires $O(\log n)$ invocations of Algorithm 3.

/* Each $c'' \in \mathsf{Comm}(u)$ has at most k copies of $m(c, u, v)$ sent by each $c' \in \mathsf{Comm}(v)$. */

for each $c'' \in \mathsf{Comm}(u)$ **do**
  if at least $\lceil (k+1)/2 \rceil$ copies of $m(c, u, v)$ are identical **then**
    c'' accepts the majority copy of $m(c, u, v)$.
  else
    Reject the message $m(c, u, v)$.

Having established the two primitives Algorithm 3 and Algorithm 4, we now present Algorithm 5 and Algorithm 6.

Algorithm 5 Parallel Agreement.

Require: For every $u \in V$, $c \in \mathsf{Comm}(u)$, and $v \in V$, the node c has a bit $b(c, u, v)$.
/* As before, there are $O(n^2 \log n)$ such bits WHP. */

Ensure: For each u and each v, the members of $\mathsf{Comm}(u)$ must reach an agreement over the bits $b(c, u, v)$ over all $c \in \mathsf{Comm}(u)$.

In parallel each pair u and v **do**
  The members of $\mathsf{Comm}(u)$ implement an $O(\text{polylog}\, n)$ round Byzantine Agreement algorithm [8] using Algorithm 4 to communicate with each other.
End parallel

▶ **Theorem 15.** *Consider the protocol described in Algorithm 2. Consider any constant $\beta < 1/3$ and a sufficiently small $\epsilon \in (0, 1/3 - \beta)$ to be fixed constants. We assume that the leader ℓ is good and the number of Byzantine nodes $B < \beta n$. The following hold with high probability.*

1. *Algorithm 2 ensures that every node $v \in V$ is represented by a committee $\mathsf{Comm}(v) \subseteq V$ of cardinality $\Theta(\log n)$. Each node participates in at most $O(\log^2 n)$ committees.*
2. *This committee structure is common knowledge amongst all nodes.*
3. *Byzantine nodes amount to a fraction of at most $\beta + \epsilon + o(1) < 1/3$ within each committee.*

Algorithm 6 Ensuring a consistent view of edges.

Require: For every $u \in V$, $c \in \mathsf{Comm}(u)$, and $v \in V$, the node c has a bit $b(c,u,v)$. If u is good then $b(c,u,v) = 1$ iff $(u,v) \in E(G)$. /* Note that c can be Byzantine and may misrepresent its bit values. */

Ensure: For every $u \in V$ and $v \in V$, the members of $\mathsf{Comm}(u)$ must reach an agreement on whether (u,v) exists. Moreover, it should be consistent with the agreement reached by $\mathsf{Comm}(v)$ regarding edge (u,v), i.e., $\mathsf{Comm}(u)$ must agree that the edge (u,v) exists iff $\mathsf{Comm}(v)$ agrees that (u,v) exists. Finally, if both u and v are good, then both $\mathsf{Comm}(u)$ and $\mathsf{Comm}(v)$ must agree that edge (u,v) exists iff $(u,v) \in E(G)$.

1: Use Algorithm 5 to ensure that for every u and every v, the members in $\mathsf{Comm}(u)$ reach an agreement (using their $b(c,u,v)$ bits). Denote the agreed bit as $a(u,v)$. Each $c \in \mathsf{Comm}(u)$ sets $a(c,u,v) \leftarrow a(u,v)$.

2: Use Algorithm 3 to ensure that for every u and every v, the members of $\mathsf{Comm}(v)$ learn $a(c,u,v)$ for every $c \in \mathsf{Comm}(u)$. /* This also means that the members of $\mathsf{Comm}(u)$ learn $a(c',v,u)$ for every $c' \in \mathsf{Comm}(v)$. */

3: For every u and every v, the members of $\mathsf{Comm}(u)$ learn $a(v,u)$ by taking majority over all $a(\cdot,v,u)$ values received in the previous step.

4: For every u and every v, the members of $\mathsf{Comm}(u)$ set their final agreement bit $f(u,v)$ to 1 iff both $a(u,v) = 1$ and $a(v,u) = 1$.

4. *For every pair of nodes u and v, $\mathsf{Comm}(u)$ has reached an agreement about whether $(u,v) \in E(G)$. Note that $\mathsf{Comm}(v)$ has also reached its own agreement. The agreements satisfy the following two properties.*

   Consistency. *$\mathsf{Comm}(u)$ reaches the agreement that $(u,v) \in E(G)$ iff $\mathsf{Comm}(v)$ reaches the agreement that $(u,v) \in E(G)$.*

   Correctness. *If u is good and $(u,v) \notin E(G)$, the committees reach the agreement that $(u,v) \notin E(G)$. Furthermore, if both u and v are good nodes, then, their committees reach the agreement that $(u,v) \in E(G)$ iff $(u,v) \in E(G)$.*

5. *Finally, the protocol terminates in $O(\text{polylog}\, n)$ rounds.*

Proof. Item (i) follows immediately from Lemma 11. Item (ii) follows from Algorithm 2 where the leader sends the $O(\log^2 n)$ bits to all the nodes and the nodes can then compute the committee structure locally at all nodes. Item (iii) is proved in item (i) of Lemma 11. Item (v) follows from Lemma 12, Lemma 13, Lemma 14, and the fact that Algorithm 6 only requires $O(\log n)$ invocations of Algorithm 3 and Algorithm 4.

For item (iv), let us focus our attention on Algorithm 6. Note that for each u the associated $\mathsf{Comm}(u)$ internally reaches agreement in Line 1 about whether $(u,v) \in E(G)$ for every other v. Then, for each pair u and v, their respective committees learn about the agreement reached by the other committee; see Line 3. In Line 4, both committees reach a consistent agreement. Moreover, they reach a common agreement that the edge $(u,v) \in E(G)$ iff both $\mathsf{Comm}(u)$ and $\mathsf{Comm}(v)$ internally reached that agreement. Thus, as long as one of them (i.e., either u or v) is good, $\mathsf{Comm}(u)$ and $\mathsf{Comm}(v)$ cannot reach a joint agreement that $(u,v) \in E(G)$ when $(u,v) \notin E(G)$. On the flip side, when both u and v are good, their committees would have reached the correct internal agreement and therefore the common agreement will also be correct. ◀

3.2 Constructing a spanning tree using committees

As mentioned earlier, we will implement a distributed version of Boruvka's algorithm by delegating the work done by each of the vertices in the network to their corresponding committees. We will start by describing an implementation of Boruvka's algorithm in the congested clique in the absence of Byzantine nodes.

The algorithm proceeds in phases, and in phase i, we have a collection of fragments $F_1, F_2, \ldots, F_r$ that are disjoint connected components of the underlying graph. Each fragment F_i has an associated fragment identifier, which is the name of one of the vertices in the fragment. We will maintain the invariant that after each phase, every vertex in the graph knows the fragment ids of every other vertex in the graph. Initially, the fragments are individual vertices, and the fragment ids are their own ids. Every vertex transmits their id to every other vertex in the clique.

Suppose that after the i^{th} phase, the fragments are $F_1, F_2, \ldots, F_r$ and that the invariant holds. Let f_v denote the fragment id of the vertex v. We will assume that the algorithm always chooses the lexicographically smallest outgoing edge from a fragment. Now, each vertex chooses the lexicographically least u in the neighborhood of v such that $f_u \neq f_v$ and broadcasts the id of u to every vertex in the clique. Now, each node can calculate the lexicographically least outgoing edge from every fragment and hence compute the new fragments and their ids. Since the number of fragments reduce by a factor of at least two every step, the number of rounds is $O(\log n)$. Notice that this algorithm is deterministic, and ends when the graph has been partitioned into connected components. Now we will see how to implement this algorithm in the presence of Byzantine nodes using committees as described in the previous subsection.

Algorithm 7 Single phase of Distributed Boruvka.

Require: Graph partitioned into fragments $F_1, F_2, \ldots, F_r$, with a fragment id associated with each fragment. Conditions of Theorem 15 hold. For each u, $\mathsf{Comm}(u)$ knows the fragment id of u which we will denote by f_u.

Ensure: Find one inter-fragment edge per fragment, and update fragment ids

1: **In parallel** $\forall c \in V$ **do**
2: $\forall u$ such that $c \in \mathsf{Comm}(u)$, send $(id(u), f_u)$ to every vertex in the clique
3: **End parallel**
4: **In parallel** $\forall c \in V$ **do**
5: $\forall u$ such that $c \in \mathsf{Comm}(u)$, find u with smallest id such that $f_u \neq f_v$ and send $(id(u), id(v))$ to every vertex in the clique.
6: **End parallel**

Correctness and Complexity. From Lemma 11, we know that, with high probability, at least half of the nodes in a committee is good. Hence, for each u, every vertex v can compute the committee of u, $\mathsf{Comm}(u)$ and use the majority value of its messages from $\mathsf{Comm}(u)$ to compute f_u. Similarly, for each u, majority of the vertices in $\mathsf{Comm}(u)$ will correctly calculate its smallest outgoing edge. Thus each vertex v, can use the majority answer from $\mathsf{Comm}(u)$ to find the outgoing edge from u. Thus, each vertex can compute the correct outgoing edge from a fragment and update the fragment ids accordingly. By Lemma 11, we know that, with high probability, each node c is present in the committee of $O(\log n)$ other vertices. Hence the round complexity of Algorithm 7 is $O(\log^2 n)$.

We now describe the final algorithm.

Algorithm 8 Connectivity Testing.

Require: Graph G in the congested clique model, and a node ℓ designated as leader known to every vertex in the graph

Ensure: Leader ℓ outputs accept (TRUE) if $G \setminus B$ is connected, and outputs reject (FALSE) if G is $2|B|+1$-far from being connected.

1: Use Algorithm 2 to obtain committees for each of the nodes in the graph that satisfies the conditions of Theorem 15.
2: Set every vertex u as a singleton fragment with fragment id as the vertex id.
3: Each node executes Algorithm 7 until there are no inter-fragment edges.
4: Leader ℓ accepts if there is a component of size at least $n - |B|$. Else ℓ rejects.

The following two lemmas prove the correctness of the algorithm. Lemma 16 shows that if indeed $G \setminus B$ is connected, then this will be observed by the leader with high probability, and Lemma 17 shows that the leader will reject, with high probability, if G is $2|B|$-far from being connected. If neither of the two conditions hold, then all the honest nodes converges on a consistent answer even though we can give no guarantee on what that answer will be.

▶ **Lemma 16.** *If the graph $G \setminus B$ is connected, then the leader returns accept with high probability in Algorithm 8.*

▶ **Lemma 17.** *Let $\beta < 1/3$ be a constant such that $|B| = \beta n$. If the graph G is $2|B|$-far from being connected, then the leader returns reject with high probability in Algorithm 8.*

We now show how to combine the results for the proof of Theorem 1.

Proof of Theorem 1. First, we will elect a leader committee of $\Theta(\log n)$ nodes such that at least 2/3-fraction of these nodes are honest. For this we use Theorem 8. Now, each leader node in this committee runs Algorithm 8 to check the connectivity of the graph. From Lemmas 17 and 16, we can conclude that if the leader node is honest, then it obtains the correct answer with high probability. To obtain the final answer we use the consensus algorithm of [8] (Theorem 10), which guarantees that the consensus obtained by the leader committee is the value of the transmitter node. Since each honest node has the wrong answer with probability at most $1/n^c$, for some constant c, by a union bound we can conclude that with high probability all honest nodes have the correct answer. Thus, repeating the consensus algorithm of [8] with each node acting as the transmitter ensures that for the majority of the values are the correct answer with high probability. ◀

4 Conclusion and Future Work

In this paper, we take a step towards robust and secure distributed graph computation by a network of nodes that can output meaningful results even in the presence of a large number of faulty or malicious nodes. This can be relevant in distributed big data applications where a large number of processors operate on a (large-scale) input and some of the processors can be faulty or malicious. We address the fundamental problem of connectivity and show that non-trivial meaningful computation can be performed in an efficient manner even in the presence of a large number of Byzantine nodes. One of the key aspects of studying graph problems in this Byzantine fault-tolerant setting is arriving at the correct problem formulation. One must contend with the fact that Byzantine nodes will enforce a "gap" between two solvable ends of a spectrum of input instances. Our work shows that we can indeed formulate such gap problems in a meaningful way. The techniques used in this paper may be useful in Byzantine distributed computation of other problems in the congested clique.

Our approach can serve as a starting point for a general technique for transforming non-Byzantine congested-clique algorithms to Byzantine ones. This work which addresses the basic connectivity problem is a step towards this larger goal which is harder. For example, generalizing our approach to even the closely related MST problem (in a weighted graph) is not immediate as it is not clear what a meaningful output for MST will be in a weighted graph in a Byzantine setting. This is an important next step and is left for future work.

Several open questions arise from our work. While the focus of this work is fast algorithms (running in polylog(n) rounds), it is also relevant to consider the message complexity of the protocol. The proposed algorithm takes $O(n^2 \operatorname{polylog}(n))$ messages in the worst case. This message complexity is essentially optimal (upto a polylog(n) factor) in the so-called KT0 model (where nodes don't have apriori information about the identities of its neighbors) since $\Omega(n^2)$ is a lower bound for connectivity testing in the congested clique even when there are no Byzantine nodes[17]. The situation is not clear if nodes have knowledge of the identities of their neighbors initially (so-called KT1 model). In this case, the question of whether $o(n^2)$ or even $O(n \operatorname{polylog} n)$ message algorithms are possible is open. Note that this is possible if there are no Byzantine nodes [17].

An interesting way to characterize performance of Byzantine protocols for more complicated problems such as connectivity is to parameterize the complexity of the performance in terms of the number of calls to Byzantine agreement which is a basic primitive. In this direction, the goal is to reduce the number of calls to agreement as much as possible.

A major next step is addressing even more challenging problems such as MST and other graph problems.

References

1 A. Andoni, Z. Song, C. Stein, Z. Wang, and P. Zhong. Parallel graph connectivity in log diameter rounds. In *Proc. IEEE FOCS*, pages 674–685, 2018.

2 John Augustine, Valerie King, Anisur Rahaman Molla, Gopal Pandurangan, and Jared Saia. Scalable and secure computation among strangers: Message-competitive byzantine protocols. In *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 31:1–31:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

3 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *J. ACM*, 64(6):40:1–40:58, 2017.

4 Michael Ben-Or, Elan Pavlov, and Vinod Vaikuntanathan. Byzantine agreement in the full-information model in o(log n) rounds. In *Proc. of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 179–186, 2006.

5 Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation - how to run sublinear algorithms in a distributed setting. In *Proc. of the 10th Theory of Cryptography Conference (TCC)*, pages 356–376, 2013.

6 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Comput.*, 32(1):41–57, 2019. doi:10.1007/s00446-018-0324-8.

7 Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in the congested clique. *Proceedings of the 39th Symposium on Principles of Distributed Computing*, July 2020. doi:10.1145/3382734.3405751.

8 Danny Dolev, Michael J. Fischer, Robert J. Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Inf. Control.*, 52(3):257–274, 1982.

9 Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three notes on distributed property testing. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 15:1–15:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.15.

10 Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

11 Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. *ACM Trans. Parallel Comput.*, 6(3):12:1–12:20, 2019. doi:10.1145/3322811.

12 R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, January 1983. doi:10.1145/357195.357200.

13 Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 19–28. ACM, 2016.

14 S. Gilbert and L. Li. How fast can you update your MST. In *Proc. ACM SPAA*, pages 531–533, 2020.

15 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

16 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

17 James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 91–100. ACM, 2015.

18 Tomasz Jurdzinski and Krzysztof Nowicki. MST in $O(1)$ rounds of congested clique. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2620–2632. SIAM, 2018.

19 H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proc. ACM SPAA*, pages 938–948, 2010.

20 Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58:18:1–18:24, July 2011.

21 Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.

22 Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 391–410. SIAM, 2015.

23 C. Konrad, S. V. Pemmaraju, T. Riaz, and P. Robinson. The complexity of symmetry breaking in massive graphs. In *Proc. DISC*, volume 146, pages 26:1–26:18, 2019.

24 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

25 S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *Proc. ACM SPAA*, pages 85–94, 2011. doi:10.1145/1989493.1989505.

26 Reut Levi, Moti Medina, and Dana Ron. Property testing of planarity in the CONGEST model. *Distributed Comput.*, 34(1):15–32, 2021. doi:10.1007/s00446-020-00382-3.

27 Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in o(log log n) communication rounds. In Arnold L. Rosenberg and Friedhelm Meyer auf der Heide, editors, *SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003)*, pages 94–100. ACM, 2003.

28 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 429–438. ACM, 2016.

29 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 405–414. ACM, 2018.

30 Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

31 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, Philadelphia, 2000.

32 Ronitt Rubinfeld and Madhu Sudan. Self-testing polynomial functions efficiently and over rational domains. In *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pages 23–32. ACM/SIAM, 1992.

33 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discret. Math.*, 8(2):223–250, 1995.

34 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

A Proof from Section 1

Proof of Theorem 3. Consider any $1 \leq b \leq n$ and $f = 2b - 1$. Suppose (for the sake of contradiction) that there is an algorithm $\mathcal{A}$ that can distinguish the following two cases with probability at least $1/2 + \epsilon$: (i) G is f-far from being connected, or (ii) $G \setminus B$ is connected. For clarity, we will use the term global algorithm to refer to the collective execution of an algorithm at all nodes and the term protocol to refer to the specific execution at an algorithm at a particular node.

We prove this theorem by constructing two graphs: (i) G_f that is f-far from being connected and (ii) G_c with a connected component of size $n - b$. We then show embeddings of these graphs on n node congested cliques with b Byzantine nodes such that the Byzantine nodes can render both embeddings indistinguishable to the global algorithm $\mathcal{A}$.

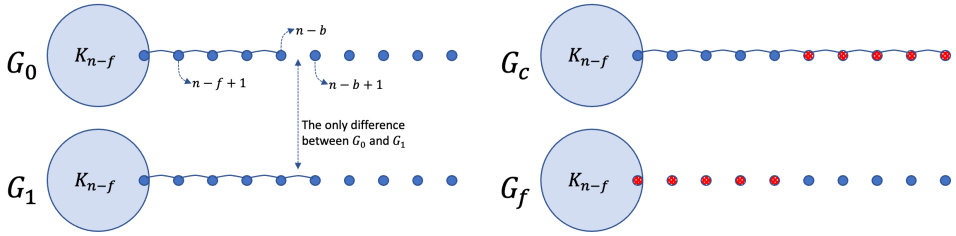

Figure 1 Graphs used in the proof of Theorem 3.

To aid us in formally showing this indistinguishability, we first construct two graphs G_0 and G_1 (that only differ in one edge) and embed them in congested cliques. See Figure 1 for schematics. Executing $\mathcal{A}$ on G_0 and G_1 will provide us with specific protocols executed by specific nodes in their respective congested cliques that can be used to precisely specify the protocols executed by Byzantine nodes in the embeddings of G_c and G_f. Both G_0 and G_1 comprise a clique of size $n - f$ (numbered 1 through $n - f$). G_0 (resp., G_1) has a "tail" comprising $b - 1$ nodes (resp., b nodes) numbered $n - f + 1$ to $n - f + b - 1$ (resp., $n - f + b$).

Specifically, the tail is a path comprising $b-1$ nodes (resp., b nodes) with one of its ends (node numbered $n-f+1$) connected to the $(n-f)$th node in the clique. Additionally, G_0 (resp., G_1) comprises $f-b+1$ (resp., $f-b$) nodes numbered $n-f+b$ to n (resp, $n-f+b+1$ to n) that are isolated. The embedding is straightforward. Vertex i is embedded into node i for all $1 \leq i \leq n$. Some arbitrary b nodes in this embedding are Byzantine. Let $\mathcal{A}_i^0$ (resp., $\mathcal{A}_i^1$) denote the protocol executed by node i in the embedding of G_0 (resp., G_1) given that i is good. Note that $\mathcal{A}$ is not required to interpret G_j, $0 \leq j \leq 1$, correctly because neither is $G_j \setminus B$ connected nor is G_j f-far from connected.

G_f comprises one clique of $n-f$ vertices (numbered 1 to $n-f$) and f isolated vertices (numbered $n-f+1$ to n); thus it is f-far from connected. See Figure 1 for a schematic representation of G_f. The clique is embedded into the first $n-f$ nodes of the congested clique and the isolated vertices are embedded into the remaining f nodes. The b Byzantine nodes are nodes $n-f, n-f+1, \ldots, n-f+b-1$. The adversary specifies the protocol executed by the Byzantine nodes i, $n-f \leq i \leq n-f+b$, to be $\mathcal{A}_i^1$. All other nodes i, by the design of $\mathcal{A}$ and the vertex embedded into them (along with incident edges matching G_g), execute their respective $\mathcal{A}_i$. Intuitively, this again produces a view of the graph that combines G_0 and G_1 with the one edge between vertices $n-b$ and $n-b+1$ under contention. The protocol $\mathcal{A}_{n-b}^1$ executed by $n-b$ will essentially operate under the existence of the edge while $\mathcal{A}_{n-b+1}$ executed by node $n-b+1$ will operate under its non-existence.

Similarly, G_c comprises one clique on $n-f$ vertices (numbered 1 to $n-f$) and a path on vertices numbered $n-f$ to n. See Figure 1 for a schematic representation of G_c. The embedding is natural, i.e., vertex i in G_c is embedded into node i in the congested clique. The b Byzantine nodes are $n-b+1$ to n and the adversary respectively executes protocol $\mathcal{A}_i^0$, $n-b+1 \leq i \leq n$, on those vertices. The rest of the vertices i, $1 \leq i \leq n-b$, by their design execute protocol $\mathcal{A}_i$. Importantly, this embedding is such that $G_c \setminus B$ is connected. Intuitively, this embedding again produces a view of the graph that combines G_0 and G_1 with the one edge between vertices $n-b$ and $n-b+1$ under contention. The protocol $\mathcal{A}_{n-b}$ executed by $n-b$ will essentially operate under the existence of the edge while $\mathcal{A}_{n-b+1}^0$ will be executed by node $n-b+1$ under its non-existence.

It is easy to verify that the protocols executed at corresponding nodes in both G_f and G_c execute the same code under the same local input. Thus, the global algorithm executed by the two embeddings of G_f and G_c are statistically identical. However, the global algorithm executed by the two embeddings must be able to distinguish with probability $1/2+\epsilon$ whether the underlying graph is G_c or G_f, thereby establishing the contradiction. ◀

B Proofs from Section 3

Proof of Lemma 11. Item (i) is easy to prove through a simple application of the Chernoff bound. Fix some node u. Recall that when $h \in \mathcal{H}$ is chosen uniformly at random, $\mathbb{P}(h(u) = v) = 1/n$. Thus, when the number of Byzantine nodes is at most βn, the $Pr(h(u)$ is Byzantine$) \leq \beta$. Since the leader chooses h_i, $1 \leq i \leq k = O(\log n)$, independently and uniformly at random from $\mathcal{H}$, we can apply Chernoff bounds to bound the number of Byzantine nodes in $\mathsf{Comm}(u)$. Let $X_i = 1$ if $h_i(u)$ is Byzantine, and 0 otherwise. Then, $X = \sum_i X_i$ will be the total number of Byzantine nodes in $\mathsf{Comm}(u)$. Clearly, $\mu = E[X] \leq \beta k$. Then, by Chernoff bounds,

$$\begin{aligned}\mathbb{P}(X \geq (\beta+\varepsilon)k) &= \mathbb{P}(X \geq (1+\varepsilon/\beta)\mu) \\ &\leq \exp\left(-\mu(\varepsilon/\beta)^2/3\right) \\ &\leq 1/\operatorname{poly}(n).\end{aligned}$$

To complete the argument that the proportion of Byzantine nodes in $\mathsf{Comm}(u)$ is at most $\beta + \varepsilon$, we also need to ensure that the committee is of cardinality at least $k - O(1)$ WHP, i.e., there aren't too many duplicated $h_i(u)$ values. This is easy to argue because, for a fixed $i \in [k]$, the probability that $h_i(u)$ is a duplicated item is at most $k/n = O(\frac{\log n}{n})$. Moreover, for $1 \leq i \leq k$, the $h_i(u)$ values are k-wise independent. Clearly then, with high probability, no more than $O(1)$ items in the committee are duplicated. Thus, we can conclude that the fraction of Byzantine nodes in the committee will be at most $(\beta+\varepsilon)k/(k-O(1)) = \beta+\varepsilon+o(1) < 1/3$ WHP when k is sufficiently large. To complete the argument, we can take the union bound over all $u \in [n]$.

To prove item (ii), we use the k-wise independence of $\mathcal{H}$ and apply Lemma 5. Let us focus on the first hash function h_1 and one node c; later we can apply the union bound over all h_i and all c. Let Y_u be 1 if $h_1(u) = c$ and 0 otherwise. If we then set $Y = \sum_u Y_u$, we get the number of committees in which c participates as the first node. Since the Y_u values are k-wise independent, we can apply Lemma 5 and get (for suitable $\delta \in \Theta(\log n)$)

$$\mathbb{P}(Y \geq (1+\delta)E[Y]) \leq \exp(-\min(k, \delta^2 t \log n)) \leq 1/\operatorname{poly}(n).$$

Item (ii) follows when we take the union bound over all h_i and all c. ◀

Proof of Lemma 12. We first bound the time and then prove correctness.

First, we need to ensure that Line number 7 takes $O(\text{polylog}(n))$ rounds WHP and this will be ensured as long as each $c \in [n]$ sends at most $O(\text{polylog}(n))$ messages. To show this, we need to bound the number of routing committees in which a node r will participate for a particular v and $c \in \mathsf{Comm}(v)$. By a straightforward application of Chernoff bounds, we get that required bound.

▶ **Lemma 18.** *Fix any v, any $c \in \mathsf{Comm}(v)$, and an arbitrary $r \in [n]$. With high probability,*

$$|\{\mathsf{RouteComm}(c, v, i) \ni r \mid i \in [N(v)]\}| \in O(\log n).$$

The requests are relayed to v and back to c only if $c \in \mathsf{Comm}(v)$. So again, the congestion in the links between c and r and between r and v will only be $O(\text{polylog}(n))$, implying that the inner parallel loop (lines 8 to 15) completes in $O(\text{polylog}(n))$ rounds. Thus, by extension, the running time for the first parallel loop is $O(\text{polylog}(n))$.

The correctness follows in a straightforward manner. If both c is good, then in expectation, fewer than β proportion of nodes will be bad in the routing committee. Thus, by a standard application of the Chernoff bound it follows that the the routing committees $\mathsf{RouteComm}(c, \cdot, \cdot)$ are all good in the sense that fewer than 1/3 proportion of nodes will be bad. Of course, bad nodes can claim to be in those routing committees. However, if v, $c \in \mathsf{Comm}(v)$, and $r \in \mathsf{RouteComm}(c, v, i)$ (for some i) are all good, then they will all follow the protocol and so each routing committee will provide a majority of correct responses. Thus, taking the union bound over all i, the proof of Lemma 12 is complete. ◀

Proof of Lemma 13. Fix a pair of nodes c and c' and consider their role as committee members. Recall that $|C(c)|$ and $|C(c')|$ are both $O(\log^2 n)$ WHP. The number of messages c sends to c' is $O(\log^4 n)$ because each $u \in C(c)$ has an associated message $m(c, u, v)$ for each $v \in C(c')$ and all these messages must traverse the link between c and c'. Moreover, no other message is passed.

The correctness follows because the messages are directly passed to all $c' \in \mathsf{Comm}(v)$. ◀

Proof of Lemma 16. If $G \setminus B$ is connected, then every non-Byzantine node is in a connected component of size at least $n - |B|$. We know that after each step of Algorithm 7, with high probability, every node knows the fragment ids of every non-Byzantine node. at the end of Algorithm 8. Therefore, at the end of the algorithm, the leader ℓ has the fragment ids of every non-Byzantine node. Since they all will lie in the same component, the leader will accept. ◀

Proof of Lemma 17. If G is $2|B|$-far from being connected, then then there are at least $2|B|+1$ connected components in the graph. Since there are only $|B|$ Byzantine nodes, there are at most $|B|$ components that contain them. Even if these Byzantine nodes claim to be connected with each other, the size of the component is less than $n-|B|$. Otherwise, since we have an additional $|B|+1$ components, the number of vertices would be $\geq n-|B|+|B|+1 > n$.

We know that the fragment ids of each honest node is known to everyone else after each run of Algorithm 7. At the end of the algorithm, the leader knows the fragment ids of every non-Byzantine node in the network. From the previous analysis we know that no component has size at least $n - |B|$, and hence, with high probability, the leader will reject this graph. ◀