Partition Detection in Byzantine Networks

Yérom-David Bromberg, Jérémie Decouchant, Manon Sourisseau, François Taïani

Partition Detection in Byzantine Networks

Yérom-David Bromberg[1], Jérémie Decouchant[2], Manon Sourisseau[1], François Taïani[1]
[1]*Univ Rennes, Inria, CNRS, IRISA*, France, [2]*TU Delft*, The Netherlands
david.bromberg@irisa.fr, j.decouchant@tudelft.nl, manon.sourisseau@inria.fr, francois.taiani@inria.fr

***Abstract*—Detecting and handling network partitions is a fundamental requirement of distributed systems. Although existing partition detection methods in arbitrary graphs tolerate unreliable networks, they either assume that all nodes are correct or that a limited number of nodes might crash. In particular, Byzantine behaviors are out of the scope of these algorithms despite Byzantine fault tolerance being an active research topic for important problems such as consensus. Moreover, Byzantine-tolerant protocols, such as broadcast or consensus, always rely on the assumption of connected networks. This paper addresses the problem of detecting partition in Byzantine networks (without connectivity assumption). We present a novel algorithm, which we call NECTAR, that safely detects partitioned and possibly partitionable networks and prove its correctness. NECTAR allows all correct nodes to detect whether a network could suffer from Byzantine nodes. We evaluate NECTAR's performance and compare it to two existing baselines using up to 100 nodes running real code, on various realistic topologies. Our results confirm that NECTAR maintains a 100% accuracy while the accuracy of the various existing baselines decreases by at least 40% as soon as one participant is Byzantine. Although NECTAR's network cost increases with the number of nodes and decreases with the network's diameter, it does not go above around 500KB in the worst cases.**

***Index Terms*—Byzantine Fault, Network Partition, Digital Signatures, Distributed Algorithms**

I. INTRODUCTION

In typical distributed systems, messages transit on communication channels that form an incomplete network. Two distant nodes only need to be connected through a path of nodes and channels to communicate. In such systems, a partition occurs when the network becomes separated into two (or more) parts that cannot communicate. Partitions can occur for various reasons, such as hardware failures, network congestion, or software issues. They can inherently lead to data loss and data inconsistency, as, for instance, commonly illustrated in the distributed database field and popularized through the P of the CAP theorem [1].

The ability to detect partitions is a key requirement for a wide range of distributed algorithms that (i) assume a connected network, (ii) need to detect specific failures (node crashes), (iii) require information about the topology (e.g., in MANETs [2]), or (iv) auto-configure systems (e.g., in IoT networks [3]). In such cases, detecting and resolving partitions quickly is essential to maintain network reliability and availability, identify and resolve underlying network issues, improve overall network performance and reduce downtime.

The partition detection problem has been well studied over the last decade [4–8]. However, while some of the existing solutions tolerate crash faults, they break down rapidly when faced with arbitrary, or even malicious behaviors. Such behaviors are often modeled as *Byzantine faults* [9]. Byzantine nodes can behave arbitrarily and, therefore, can act in the worst possible way for the correctness of the system.

Although Byzantine faults have been widely studied in a broad range of contexts [9–17], almost all existing works assume a connected network.[1] In particular, to the best of our knowledge, no work has studied the problem of detecting partitions in *arbitrary* network topologies in the presence of Byzantine nodes.

Detecting a network partition under a Byzantine fault model is however far from trivial. From an operational point of view, correct nodes typically need to decide whether their communication with other correct nodes is guaranteed despite Byzantine nodes. Although simple to state, such a property can be difficult to assess in a distributed setting, in particular if the graph of communication is not immediately known to nodes. This is because the position of Byzantine nodes is not known to correct nodes, and because Byzantine nodes may behave correctly during the partition detection algorithm (thus hiding the danger they represent), and only disrupt communication at some critical later point if they happen to be located at key positions in the network.

To apprehend this difficulty, we introduce the operational notion of *t-Byzantine partitionability* and link it to the vertex-connectivity of the underlying communication graph. We then formally specify the problem of *Byzantine-resilient network partition detection*, and introduce NECTAR, the first algorithm to solve this problem in a synchronous communication model with signatures, along with its proof of correctness. Contrary to earlier Byzantine algorithms on arbitrary graphs [11, 18], NECTAR's does not require any minimal connectivity value and conserves its safety and liveness properties on any graph.

Contributions. We make the following contributions:

- We formally define a new problem, *Byzantine-resilient network partition detection*, and present NECTAR, a distributed algorithm that solves this problem on any arbitrary graph in the presence of Byzantine nodes.
- We formally prove that NECTAR solves Byzantine-resilient network partition detection .
- We have implemented a prototype of NECTAR in C++ on top of a real network stack on real machines.

[1]The work of Augustine, Molla, Pandurangan, and Vasudev [17] is the only exception we know of. It is, however, limited to *congested cliques* (a type of fully-connected communication networks).

- We thoroughly evaluate our approach *via* an in-depth experimental evaluation, comparing both network costs and Byzantine reliability to that of two baselines.
- We demonstrate that state-of-the-art algorithms lose around 40% of accuracy in the presence of a single Byzantine node, while our solution remains correct, even in the presence of multiple Byzantine faults. This robustness comes at a reasonable network cost, which does not exceed 500 KB per node, in the worst case, for a 100-node system.

The rest of this paper is organized as follows. Section II describes our system model. Section III recalls the definition of a partitioned network, defines t-Byzantine partitionable networks, and Byzantine-resilient partition detection algorithms. Section IV presents NECTAR, our network partition detection algorithm, and proves its properties. Section V evaluates NECTAR's performance and compares it to two non-Byzantine-resilient baselines. Section VI discusses the related work. Finally, Section VII concludes this paper.

II. System Model

We consider a set $\Pi = \{p_1, p_2, ..., p_n\}$ of n processes, each identified by a unique ID. All processes know the total number of processes, $\mathbf{card}(\Pi) = n$, and their ID. Processes are interconnected by a network represented by a static undirected graph $G = (V, E)$ where $V = \Pi$ and where E contains the communication channels between pairs of nodes. We use k to denote the vertex connectivity of G. For simplicity, we assimilate each node of G with the process it hosts, and will use the words '*process*' and '*node*' interchangeably.

Two nodes can directly communicate through a communication channel if and only if they are connected by an edge in G. Otherwise, they must rely on other nodes to relay their messages. We assume that communication channels are reliable, and that the network is synchronous. In other words, there is a bound ΔT so that messages sent at a time T_0 are always received before time $T_1 = T_0 + \Delta T$. We assume that the local processing time of nodes is negligible compared to communication delays.

Nodes can sign and authenticate messages using an asymmetric digital signature scheme. We note $\sigma_i(msg)$ a message msg signed by node i. Our protocol leverages chained signatures. For example, $\sigma_j(\sigma_i(msg))$ is a chain signature of message msg that allows a node to extract and verify $\sigma_i(msg)$ and $\sigma_j(\sigma_i(msg))$.

The *neighborhood* of a node i—noted $\Gamma(i)$—is the set of nodes with which it can directly communicate in G, i.e. $\Gamma(i) = \{j \in V \mid (i,j) \in E\}$. When $j \in \Gamma(i)$, we say that j is a *neighbor* of i.

We assume that the system contains up to t Byzantine nodes. Byzantine nodes may deviate arbitrarily from their specified protocol, e.g., they may drop, modify, or inject messages at any time. Following the standard Byzantine fault model [9, 11, 18], Byzantine nodes may not, however, violate network assumptions, such as synchrony (belated messages from Byzantine nodes are just ignored by correct nodes) or reliable channels. In particular, Byzantine nodes cannot prevent two correct neighbors from communicating with each other. Byzantine nodes cannot forge signatures, ensuring the authenticity and integrity of messages forwarded by correct nodes. They cannot spawn new nodes or generate new identities, eliminating Sybil attacks [19].

Nodes do not know G, the underlying network's topology, but know their individual neighborhood $\Gamma(i)$ (either because it was statically configured at set-up, or because it is provided by the underlying network stack, which we assume immune to Byzantine interference). Each node p_i has further access to a cryptographic *proof of neighborhood* (noted $proof_{p_i,p_j}$) for each of its neighbors $p_j \in \Gamma(i)$. Byzantine nodes cannot forge $proof_{p_i,p_j}$ if it involves at least one correct node. Byzantine nodes may however forge proofs of neighborhood between Byzantine processes.

III. Byzantine Partition Detection

This section introduces the notion of t-Byzantine partitionable network, and formally define the problem of Byzantine-resilient network partition detection.

A. Network partition

Informally, a network partition corresponds to the impossibility for pairs of nodes in a network to communicate, even if they rely on intermediary nodes to relay their messages. More formally, this can be defined as follows:

Definition 1 (Network partition)**.** *A communication network $G = (V, E)$ is partitioned if there is a partition $P = \{V_1, \cdots, V_k\}$ of V in $k \geq 2$ subsets such that $\forall (u, v) \in V_i \times V_{j \neq i}$, $(u, v) \notin E$.*

Unfortunately, whether Def. 1 is satisfied does not characterize the impossibility for pairs of correct nodes to communicate reliably in the presence of Byzantine processes since Byzantine nodes might be able to prevent correct nodes from exchanging messages in a non-partitioned graph. We introduce the notion of t-Byzantine partitionability to capture this notion.

B. t-Byzantine Partitionability

We say that a graph is *t-Byzantine partitionable* if it contains two correct nodes that might be unable to exchange messages if t other nodes are Byzantine. More formally, t-Byzantine partitionability is defined as follows.

Definition 2 (t-Byzantine partitionable graph)**.** *A communication graph G is t-Byzantine partitionable if all algorithms executing on G have at least one execution in which at least one pair of correct nodes cannot exchange messages.*

In practice, the notion of t-Byzantine partitionable network does not imply that pairs of correct nodes can never exchange messages. Correct nodes might still be able to communicate if strictly less than t nodes are Byzantine, or if the Byzantine nodes continue to relay messages. This notion is directly related to the vertex connectivity of the graph G, as stated by the following theorem and its corollary.

Theorem 1. *A network $G = (V, E)$ is t-Byzantine partitionable iff there is a set $V_b \subset V$ of t nodes or less such that the subgraph induced by $V \setminus V_b$ is partitioned.*

Proof. • Let us assume that a network $G = (V, E)$ is $t-$Byzantine partitionable, and consider a simple algorithm $\mathcal{A}$ in which, in round 1, every node sends its signed identifier to all its neighbors. Then, starting in round $r \geq 2$, each node retransmits in round $r + 1$ all the messages received in round r. By definition of t-Byzantine partitionability, there exists an execution $E_{\mathcal{A}}$ of $\mathcal{A}$ in which at least one pair of correct nodes (v_i, v_j) cannot exchange messages, in particular v_j never receives v_i's signed identifier. Let us note $V_b^{\mathcal{A}}$ the set of Byzantine nodes in $E_{\mathcal{A}}$. Because all nodes in $V \setminus V_b^{\mathcal{A}}$ are correct, they execute $\mathcal{A}$ faithfully. The fact that v_j never receives v_i's signed identifier implies there is no path between v_i and v_j in $V \setminus V_b^{\mathcal{A}}$, and therefore that $V \setminus V_b^{\mathcal{A}}$ is partitioned.

• Turning to the reverse implication, consider a network $G = (V, E)$ such that there is a set $V_b \subset V$ of t nodes such that the subgraph G_b induced by $V \setminus V_b$ is partitioned. Consider an algorithm $\mathcal{A}$ executing on G. Consider an execution $E_{\mathcal{A}}$ of $\mathcal{A}$ in which all nodes in V_b are Byzantine (which is possible as $|V_b| = t$) and all nodes in $V \setminus V_b$ are correct. Assume all Byzantine nodes drop all messages they receive. As a result, in $E_{\mathcal{A}}$, the messages of $\mathcal{A}$ sent by correct nodes may only travel on the edges of G_b, the graph induced by $V \setminus V_b$. As G_b is partitioned, there exist two correct nodes in $V \setminus V_b$ that cannot exchange messages in $E_{\mathcal{A}}$. □

Since in a given execution the identity of the t possible Byzantine processes is unknown, using Theorem 1 to test whether a graph G is t-Byzantine partitionable requires that every possible set V_b of t processes be tested. Such a test directly translates into a condition on the *vertex-connectivity* of the graph G, defined as the size of the smallest vertex subset of G that partitions G.

This translates into the following condition:

Corollary 1. *A network $G = (V, E)$ is t-Byzantine partitionable iff. its vertex connectivity is lower than or equal to t.*

Proof. This directly follows from the Theorem 1 and the definition of the vertex connectivity of a graph. □

In a graph with connectivity k larger than t, the subgraph of correct nodes remains connected no matter how the t Byzantine nodes are placed. As a result, correct nodes can continue to exchange messages (possibly indirectly) independently of the Byzantine nodes' behavior. For instance, the graph in Fig. 1a is 2-connected. With $t = 1$, a single Byzantine node cannot prevent the remaining correct nodes from communicating with each other, whichever its position in the graph.

By contrast, $k \leq t$ does not necessarily imply that correct nodes cannot communicate. The disruption that Byzantine nodes may cause will depend in this case on their position in the graph. However, $k \leq t$ implies that at least one such placement exists in which the t Byzantine nodes can prevent correct nodes from communicating with each other. In the star graph shown in Fig. 1b, if $t = 1$, a Byzantine node will prevent correct nodes from communicating with each other only if it is placed in the center position.

(a) (b)

Fig. 1. (a): A graph that is not 1-Byzantine partitionable. No matter the placement of a Byzantine node, the subgraph of correct nodes remains connected. (b): A 1-Byzantine-partitionable graph. If the red node is Byzantine, then the subgraph of the correct nodes is partitioned.

C. Byzantine network partition detection

Ideally, a partition detection algorithm should ensure that all correct nodes reach the same correct conclusion: either that they will remain able to communicate with one another, independently of the behavior of Byzantine nodes, or that they will not. Unfortunately, the boundary between these two situations is not as clear-cut as it seems. This is because t-Byzantine partitionability (Definition 2) captures the existence of a worst case where at least one placement of Byzantine nodes can disrupt the communication between correct nodes. This worst case is not guaranteed and only occurs if Byzantine nodes form a vertex cut of the graph G. If they do not, even a t-Byzantine partitionable graph G will allow correct nodes to communicate with one another independently of Byzantine behaviors.

Because correct nodes do not know which nodes are Byzantine, and Byzantine nodes might pretend to be correct (at least while the partition detection algorithm executes), correct nodes cannot distinguish a favorable placement from an unfavorable one. In a Byzantine context, a partition detection algorithm must, therefore, foresee two outcomes:

- NOT_PARTITIONABLE. No placement of Byzantine nodes can disconnect correct nodes.
- PARTITIONABLE. Byzantine nodes might be able to disconnect correct nodes (but this is not certain).

t-Byzantine partitionability (and vertex connectivity) might appear as a natural metric to characterize these two cases. Unfortunately, correct nodes initially do not know G, and must instead collaborate to gather information regarding G. Byzantine nodes can disrupt this process, and distort how correct nodes perceive G (by not communicating some edges, for example). As a result, if a correct process observes a connectivity of t, it cannot distinguish between the following two situations: (i) the network connectivity is indeed equal to t and the network is t-Byzantine partitionable; and (ii) the network connectivity is in fact higher than t but some Byzantine nodes omitted to send some messages, thus corrupting this correct node's perception. In this case, the correct node can only conclude that the network *might* be t-Byzantine partitionable. One direct consequence of this scenario is that correct nodes might conclude to a partitionable graph, where the real connectivity is great enough to avoid partition.

D. Formal Specification

In the light of the previous discussion, we define a network partition detection algorithm as a distributed algorithm that provides each node with a call-back decide() that returns one of two possible values: NOT_PARTITIONABLE, or PARTITIONABLE. Correct processes execute decide() once[2]. The algorithm is further parameterized by a known threshold $k_0 > t$ that characterizes its tendency to make conservative decisions.

Definition 3 (t-Byzantine-resilient, k_0-sensitive network partition detection)**.** *We say that a network partition detection algorithm $\mathcal{A}$ is t-Byzantine-resilient and k_0-sensitive if it satisfies the following properties for any communication graph G and placement of Byzantine nodes V_b (with $|V_b| \leq t$):*

- ***Termination.*** *All correct nodes decide within a bounded amount of time.*
- ***Agreement.*** *All correct nodes decide the same value.*
- ***Safety.*** *If Byzantine nodes can effectively prevent communication between correct nodes (formally, if V_b is a vertex cut of G) then no correct process ever decides* NOT_PARTITIONABLE.
- ***k_0-Sensitivity.*** *If G's connectivity is higher or equal to k_0, then all correct nodes decide* NOT_PARTITIONABLE.

The algorithm we present in the next section, NECTAR, fulfills the properties of Definition 3 with $k_0 = 2t$. It also provides an additional Boolean output, confirmed, that indicates that a correct node has detected an actual partition, i.e., that Byzantine nodes can effectively filter/cut off communications between correct nodes. Formally, this additional output fulfills the following property:

- **Validity.** If a correct node computes confirmed $=$ True, then V_b is a vertex cut of the graph G.

IV. NECTAR: NEIGHBORS EXPLORING CONNECTIONS TOWARD ADVERSARY RESILIENCE

NECTAR solves the t-Byzantine-resilient, $2t$-sensitive network partition detection problem that we have just defined. NECTAR does not require nodes to know the underlying network topology, except for the identity of their immediate neighbors (in the form of neighborhood proofs). To the best of our knowledge, it is the first algorithm to solve the problem of network partition detection in arbitrary graphs in a Byzantine context.

A. Overview

Inputs/Output. NECTAR executed at a process p_i takes 4 parameters as input: (i) the number of nodes in the system (n); (ii) the maximum number of Byzantine nodes (t); (iii) the neighborhood $\Gamma(i)$ of the local node i; (iv) a proof of neighborhood for each of its neighbors ($proof_{i,j}$ for $j \in \Gamma(i)$).

[2]The specification and the algorithm we present assume a static graph and are therefore *one-shot*. In practical cases, the connectivity graph might, however, evolve over time. In such cases, we assume that the graph remains static long enough for the algorithm to execute.

A *proof of neighborhood* $proof_{p_i,p_j}$ is a cryptographic object used by p_i to declare an edge with process p_j that cannot be forged as soon as either p_i or p_j is correct.

NECTAR's output is one of two possible decisions: NOT_PARTITIONABLE and PARTITIONABLE. It provides moreover an additional indicative Boolean output, confirmed. The NOT_PARTITIONABLE value corresponds to a situation where the graph is not partitioned and cannot be partitioned by the t Byzantine nodes. PARTITIONABLE corresponds to the case where the node evaluates that the graph is connected but that its connectivity appears lower than t, i.e., the graph is t-Byzantine partitionable. The confirmed boolean identifies a partitioned network where some nodes are unreachable.

Intuition. The key idea of NECTAR is to make nodes estimate the vertex-connectivity of the network to detect whether the network is partitioned or could be partitioned by t Byzantine processes not relaying messages. To do so, nodes disseminate their edges in signed messages and evaluate the size and connectivity of the graph they obtained after a limited number of rounds. Focusing on communication edges instead of trying directly to communicate with every node is one way to avoid correct nodes to conclude to a partition situation, when faulty nodes act as crashed ones, for example, even if the communication network is actually connected. It, however, strongly relies on the hypothesis that each node knows its neighborhood (and can prove it). The use of signature chains, with a length corresponding to the number of rounds, limits the impact of Byzantine nodes but also ensures that all correct nodes will reach the same conclusion. Sec IV-C gives the intuition on how NECTAR satisfies the properties of Def 3.

B. Pseudocode

Alg. 1 details NECTAR's pseudocode, which we discuss in the following.

Initialization. Initially, each node keeps in memory an adjacency matrix that will contain all the edges it discovers during the algorithm's execution. This adjacency matrix is initialized as follows (ll. 1-4): for each neighbor, the matrix contains locally-generated proofs of the neighborhood. For example, if i and j are neighbors, then i will initialize G_i such that $G_i[i,j] = proof_{i,j}$. Each node also initializes empty buffers $(to\_be\_sent\_i)_i$, that are necessary for the synchronous communication phase.

Edge Propagation phase. This phase comes right after the initializing phase (ll. 5-15). Nodes communicate synchronously during $n-1$ communication rounds. This phase allows nodes to share their knowledge of the graph, i.e., transmit the edges they know to their neighbors. In particular, nodes send their neighborhood (with signed proofs) during the first round (ll. 6-8). When a node receives a signed message, it verifies it and relays it with its signature, which generates signature chains, to its neighborhood at the beginning of the next round (ll. 10-12). Upon reception of a signature chain

(ll. 13-15), a process checks its correctness, but also its length (l. 14). A correct execution implies that the length of the signatures chain is always equivalent to the current round number. This verification prevents Byzantine nodes from transmitting late messages. To avoid over-flooding, nodes make sure not to resend an edge they have already sent (l. 14).

Decision phase. After $n-1$ rounds, all correct nodes have the same adjacency matrix and are able to compute if all nodes are reachable and if the vertex-connectivity is greater than t, to determine if the graph is partitioned or t-Byzantine partitionable (ll. 16-23). Note that to accurately evaluate the topology, the number of rounds to use R should be larger than the graph diameter. Since we assume that nodes do not know the topology, $n-1$ rounds are the lowest R value one can use (the worst case being the chain topology). Choosing a different value for R does not change the message complexity of NECTAR since no node will learn a new edge after the round that corresponds to the graph diameter, and all correct nodes will stay silent in the following rounds.

Impact of Byzantine deviations. NECTAR limits Byzantine nodes' ability to lie about their neighborhood or create knowledge disparities among correct nodes. However, it cannot compel Byzantine processes to share their own neighborhood correctly. In particular, edges that connect two Byzantine nodes might never be discovered, which might decrease the graph's vertex connectivity below t. In this case, correct nodes will decide that the graph is PARTITIONABLE, while it is not in reality. However, pairs of Byzantine nodes that declare fictitious edges connecting them are not an issue because these edges will never increase the vertex-connectivity above t if the subgraph of correct nodes is partitioned. In this case, the correct nodes will decide that the graph is PARTITIONABLE.

C. Agreement, Safety, and Sensitivity Properties

We now give the intuition on how NECTAR enforces the properties of Def. 3, which are more formally proven in Sec. IV-D.

The values that correct processes decide depend on the graph connectivity and on the connectivity of the correct process subgraph. We identify three main cases:

- If $2t \leq k$ (case 1), then all correct nodes decide NOT_PARTITIONABLE.
- The case $k \leq t$ is divided in two subcases. First, if $0 < k \leq t$ and the subgraph of correct nodes is connected (case 2.1), then all correct nodes decide PARTITIONABLE (with confirmed $=$ True if they detect a real communication issue, confirmed $=$ False otherwise), or NOT_PARTITIONABLE. Note that the NOT_PARTITIONABLE value can be decided even if $k \leq$t, but is not an issue, as discussed previously about the impact of Byzantine deviations in Sec IV. Second, if $0 < k \leq t$ and the correct nodes subgraph is not connected (case 2.2), then all correct nodes decide PARTITIONABLE with confirmed $=$ True if they detect a real communication issue, confirmed $=$ False otherwise). In this case, correct nodes do not necessarily reach the same conclusion about confirmed boolean, but they all decide that the network is PARTITIONABLE. For those two cases, note that if $k = 0$, then all correct nodes decide PARTITIONABLE (Byzantine nodes cannot increase the connectivity more than t). If we consider that Byzantine nodes can exchange messages through another network, correct nodes, however, do not necessarily compute confirmed $=$ True
- If $t < k < 2t$ (case 3), then either all correct nodes decide PARTITIONABLE (Byzantine nodes might not share some edges, that might decrease the perceived connectivity below t) or they all decide NOT_PARTITIONABLE. However, correct nodes compute confirmed boolean to False

Input : $n, t, \Gamma(i), (proof_{i,j})_{j\in\Gamma(i)}$; ▷ *for node* i

Output : decide { NOT_PARTITIONABLE or PARTITIONABLE

Initialising G_i

1 $G_i = matrix(n, n)$ ▷ *empty matrix* ;
2 **foreach** $neighbor \in \Gamma(i)$ **do**
3 $G_i[i, neighbor] \leftarrow proof_{i,neighbor}$
4 $to\_be\_sent_1 = \emptyset$;

Edge Propagation

5 **In each synchronous round** $R \in [1, n-1]$ **do**
6 **if** $R = 1$ **then**
7 **foreach** $neighbor \in \Gamma(i)$ **do**
8 send $\{\sigma_i(proof_{i,j})\}_{j\in\Gamma_i}$ to $neighbor$
9 **else**
10 **foreach** $(msg, k) \in to\_be\_sent_R$ **do**
11 **foreach** $neighbor \in \Gamma(i) \setminus k$ **do**
12 send $\sigma_i(msg)$ to $neighbor$
13 **when** $msg = \sigma_k(\sigma_x(\sigma_y(...., \sigma_u(proof_{u,v}))))$ **from** k **is received do**
▷ *Invalid messages are ignored*
14 **if** lengthSign$(msg) = R$ *and* $G_i[u][v] \neq nil$ **then**
15 add (msg, p_k) to $to\_be\_sent_{R+1}$
$G_i[u][v] = proof_{u,v}$

Decision

16 $r =$ DetectReachableNode(G_i) ;
17 $k =$ VertexConnectivity(G_i) ;
18 **if** $k > t$ *and* $r = n$ **then**
19 confirmed $=$ False ;
20 decide(NOT_PARTITIONABLE)
21 **else**
22 **if** $r = n$ **then** confirmed $=$ False;
23 **else** confirmed $=$ True;
24 decide(PARTITIONABLE) ;

Algorithm 1: NECTAR's pseudocode

D. Proof of correctness

We divide our proof in four lemmas, that refer to the three cases of the previous section (Sec. IV-C). We note $G_{i,r}$ the state of the adjacency matrix of node i at the end of round

r ($r = 0$ corresponds to the initial state of G_i), B the set of Byzantine nodes, and C the set of correct nodes.

Lemma 1. *(case 1) If the graph is (at least) $2t$-connected, then all correct nodes decide* NOT_PARTITIONABLE.

Proof. We suppose that the graph is $2t$-connected. First, we show that every correct node computes $r = n$. As we suppose that the graph is $2t$-connected, by definition, the graph is therefore also $t+1$-connected. Due to Menger's theorem [20], there exist at least $t + 1$ vertex independent paths between every pair of nodes. Because there are at most t Byzantine nodes, it implies that every Byzantine node is the neighbor of a correct node. With our notations, this property can be written as:

$$\forall b \in B,\ \exists i \in C : G_{i,0}[i, b] = \sigma_i(proof_{i,b}) \tag{1}$$

We now show that for two correct nodes, c_1 and c_2, at the end of $n - 1$ rounds, we have:

$$\forall c_1, c_2 \in C,\ G_{c_1,1} \subseteq G_{c_2,n-1} \tag{2}$$

Using Menger's theorem [20], because the graph is at least $t + 1$-connected, there is a path $(p_1, ..., p_m)$ of correct nodes in G such that $c_1 = p_1$, $c_2 = p_m$ and $m \leq n$. We can trivially show by induction on k ($k \in [1, m-1]$), that $G_{p_1,0} \subseteq G_{p_{1+k},k}$, because the neighborhood of node c_1 propagates from node to node in each round (assuming synchronous communication) following the correct node path. For $k = m - 1$, we have thus $G_{c_1,0} \subseteq G_{c_2,m-1}$.

A corollary of Equations 1 and 2 is:

$$\forall b \in B \cup C,\ \forall i \in C,\ \exists k \in C : G_{i,n-1}[k, b] = \sigma_k(proof_{k,b}) \tag{3}$$

This corollary means that each correct node i sees every node b as reachable, and therefore computes $r = n$.

Second, we show that every correct node computes $k > t$.

A direct corollary of Equation 3 is that each node sees every edge between two correct nodes (if $b \in C$) and every edge between correct nodes and Byzantine nodes (if $b \in B$). Because we suppose that the graph is $2t$-connected, even by removing the edges between Byzantine nodes, which can drop the connectivity by at most $t - 1$, every node computes (at least) a $t + 1$-connectivity. Then every node decides value NOT_PARTITIONABLE.

□

Lemma 2. *(case 2.1 and 3) If the subgraph of correct nodes is connected, then all correct nodes decide the same decision.*

Proof. Let us prove that, at the end of round $n - 1$, every correct node computes the same graph topology, noted G_f:

$$\forall i, j \in C,\ G_{i,n-1} = G_{j,n-1} = G_f \tag{4}$$

Because the subgraph of correct nodes is connected, we have:

$$\forall i, j \in C,\ \exists k \in [1, n-1],\ \forall u \in \Gamma(i),\ G_{j,k}[i, u] = proof_{i,u} \tag{5}$$

This comes from the fact that each node propagates its neighborhood to every other reachable node. Since the subgraph of correct nodes is connected, every correct node will receive this neighborhood during the $n - 1$ rounds.

Each message sent by a correct node will thus be received by every other correct node. Thus, we have:

$$\forall i, j \in C,\ G_{i,0} \subseteq G_{j,n-1} \tag{6}$$

We now show that $\forall i, j \in C,\ \forall u \in B \cup C,\ \forall b \in B,\ \forall k \in [1, n-1]$,

$$G_{i,k}[b, u] = proof_{b,u} \Rightarrow \begin{cases} \exists k' \in [1, n-1], \\ \quad G_{j,k'}[b, u] = proof_{b,u} \end{cases} \tag{7}$$

Intuitively, if a correct node receives (and accepts) a message from a Byzantine node during the $n - 1$ rounds, then every correct node will receive this same message.

We use the same argument as Dolev and Strong [18]: if a message msg=$\sigma_r(\sigma_{r-1}(\sigma_{r-2}(...\sigma_1(proof_{b,u}))))$ is received by a correct process for the first time in round $r \leq n - 1$ (meaning that no other correct process has received msg in a round $r' \in [1, r-1]$), then all the processes that have signed msg in the rounds $[1, r]$ must be faulty, and $r \leq t - 1$. The correct node that is the first to receive the message msg will propagate a signed message (i.e., will add its signature to the chain and forward it to its neighbors) containing msg in round $r + 1$, and all correct processes will receive msg at the latest by round $r + 1 + d - 1 \leq t + d - 1$ where d is the diameter of the graph of correct nodes ($d - 1$ rounds is thus sufficient for a message to travel among the graph of correct nodes). As $d < n - t$ ($n - t$ is the number of correct nodes), all correct nodes receive a signatures chain containing msg by round $n - 1$.

One can note that this earlier argument makes the assumption that the system contains exactly t Byzantine nodes. If the number of effective Byzantine nodes is lower than t, the same reasoning holds.

So finally,

$$\forall i, j \in C,\ G_{i,n-1} = G_{j,n-1} = G_f \tag{8}$$

At the end of the $n - 1$ rounds, all correct nodes will have the same view of the graph topology ($\forall i \in C, G_{i,n-1} = G_f$). Thus, every correct node will compute the same r and the same k, leading them to reach the same decision. □

Lemma 3. *(case 2.2) If the subgraph of correct nodes is disconnected, then all correct nodes will decide* PARTITIONABLE.

Proof. Let us assume that the subgraph of correct nodes is disconnected and that a correct node decides NOT_PARTITIONABLE. It means that this correct node has computed a vertex connectivity of at least $t + 1$. Thus, it means that it received signatures from every node through at least $t + 1$ vertex-disjoint paths. Thus, there is a path between this node and every other correct node that is free of Byzantine nodes, which means that the subgraph of correct nodes is connected and contradicts our assumption. □

Theorem 2. *Algorithm 1 is a t-Byzantine-resilient network partition detection algorithm (defined in Def. 3).*

Proof. Alg. 1 satisfies the termination property due to the network synchrony hypothesis. Agreement is ensured by Lemma 2 and Lemma 3. Safety is ensured by Lemma 3. Finally, NECTAR is $2t$-sensitive, according to Lemma 1. Regarding the validity property, if a node computes confirmed = True, it has in particular, computed $r \neq n$ (Alg. 1 (ll. 16-23)). There are thus two cases: first, if the subgraph of correct nodes is disconnected, then V_b is directly a vertex cut of G. Second, if the subgraph of correct nodes is connected, then according to Lemma 2, every correct node computes $r \neq n$. It thus means that, according to Eq 5, there exists $b_0 \in B$, such that for every correct node $c \in C$, b_0 is not in $\Gamma(c)$, i.e., V_b is a vertex cut of G. □

E. Communication Complexity.

According to the pseudo-code of Alg. 1, we can compute the message complexity as follows. Each node has to forward a unique message for each network edge, one for each of its neighbors. In the worst case (fully connected graph), the complexity is thus in $O(n^4)$. A key aspect is that the complexity highly depends on the network topology. The more edges the graph has, the higher the global communication cost is. The communication cost can also be very disparate through nodes since the complexity for each node depends on the size of its neighborhood. One can also note that the data sent cost through the algorithm might vary a lot: the lower the graph's diameter is, the sooner the nodes will have exchanged their neighborhood information. It might thus happen that in the last rounds of the algorithm, every correct node stays silent because they have already discovered all the edges of the graph.

V. EVALUATION

We compare NECTAR experimentally to two baselines on different families of network topologies in terms of network cost and resilience to Byzantine behaviors.

A. Baselines

To the best of our knowledge, there is no existing Byzantine tolerant partition detection algorithm in the literature that is compliant with any topologies. We, however, compare NECTAR to MindTheGap (MtG) [6], an efficient network partition detection solution. Processes in MtG flood a list of reachable nodes to each other. Nodes keep in memory a list of reachable nodes (that only contains themselves initially), and send regularly this list to their neighbors, during a fixed period of time (an *epoch*). When receiving a list of neighbors, nodes can actualize their own list of reachable nodes. MtG has a low network consumption because it uses Bloom filters to represent a list of process IDs.

MtG mechanisms are easily corruptible by a single Byzantine process (as explained in Sec. V-D). We thus decided to also consider a strengthened version of MtG as a second baseline, where MtG's Bloom filters are replaced by a list of signed process IDs. To minimize the increased network cost associated to this modification, we made sure that nodes only send a given signed ID once to their neighbors per epoch. In the following, we call MtGv2 this variant of MtG.

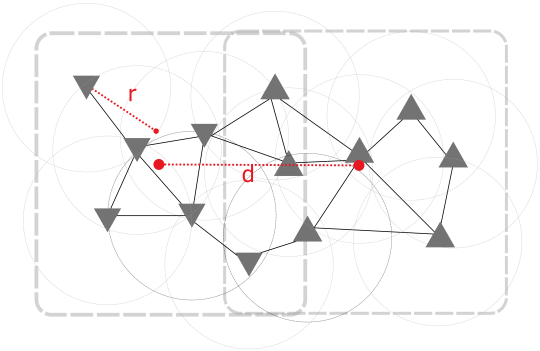

Fig. 2. An example of our drone scenario with random graph. Two scatters of points are generated. d is the distance between the barycenters of the scatters, and r is the communication scope.

B. Experimental setup

We implemented all protocols in `C++` and used the `salticidae` networking library[3]. We use the ECDSA signature scheme [21]. We ran our experiments on a Dell PowerEdge R640 server (2x Intel(R) Xeon(R) Gold 6136 CPU @ 3.00GHz, 12 cores/processor, hyper-threading, 188GB of RAM). Our deployment code uses one `Docker` container per process [22].

To study the network cost of NECTAR, we consider two families of graph topology:

- (i) Realistic connectivity-dependent topologies, such as those considered by Bonomi, Farina, and Tixeuil [23], since connectivity is a key aspect of our approach.
- (ii) Random graphs that model a simple drone network.

Bonomi, Farina and Tixeuil [23] highlight several class of graphs, such that:

- $k-$regular $k-$connected graphs [24]. Regular graphs ensure that the graph's connectivity is exactly k (with the minimum number of edges) and that each node has exactly k neighbors.
- $k-$pasted-tree and $k-$diamond graphs [25]. Those topologies are *Logarithmic Harary Graph*, built to have interesting properties for fault-tolerance and suit message flooding communication protocols.
- Generalized and Multipartite Wheel graphs [23]. Those topologies are the worst-case scenarios while considering Byzantine faults: Byzantine nodes might compose a clique while it might have only one (generalized wheel) or few (multipartite wheel) path(s) that link all correct nodes (and thus increase latency).

For the drone scenario, we create random graphs by generating random nodes in a 2D space, and a scope parameter decides edges: if two nodes are close enough (i.e., their distance is lower than $radius$), then we add an edge between them. Those nodes are randomly generated around two barycenters. In the

[3]https://github.com/Determinant/salticidae

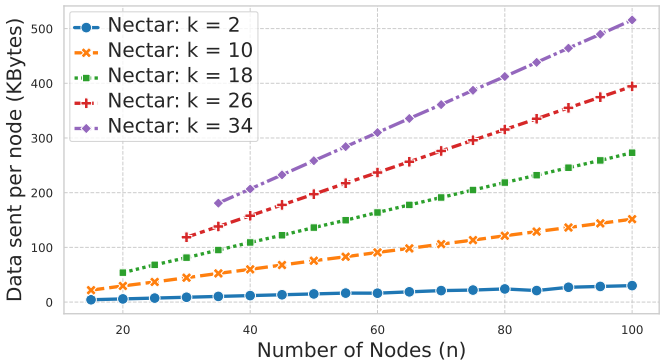

Fig. 3. Data sent per node (in KB), depending on the number of nodes (n), for different vertex-connectivity (k) in $k-$regular graphs, for NECTAR

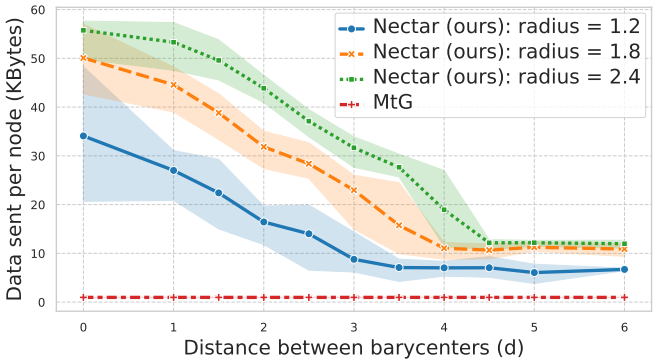

Fig. 4. Data sent per node (in KB), depending on the distance between barycenters (d), for different values of communication scope ($radius$), in the drone scenario for NECTAR. The red curve is MtG [6], whose performance does not depend on d nor $radius$.

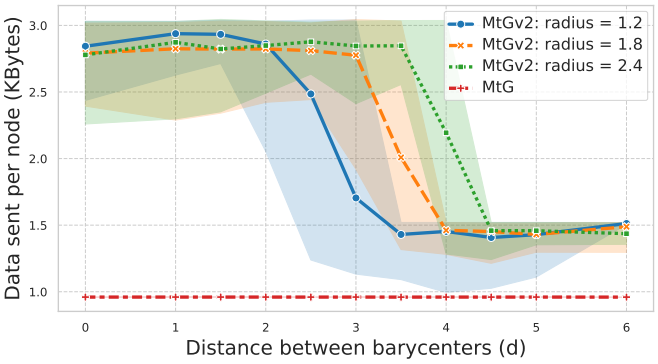

Fig. 5. Data sent per node (in KB), depending on the distance between barycenters (d), for different values of communication scope ($radius$), in the drone scenario for MtGv2.

following, we vary the distance between barycenters (noted d), the scope parameter ($radius$), and the number of nodes (n). Fig 2 is an example of our scenario, which aims to model a drone network, where two drone scatters are moving away or approaching in space. For each situation, we run 50 times the experimentation and report average results. Error intervals correspond to a confidence interval of 95% we obtained during our experimentation.

We finally selected a number of nodes to act as Byzantine nodes to study the Byzantine resilience of the tested algorithms. The tested Byzantine behaviors are discussed in Sec. V-D.

C. Network cost

Testing impact of connectivity. We discuss in this section of the performance of NECTAR on the topologies highlighted by Bonomi and al. [23], while varying the connectivity parameter. Fig. 3 shows the data sent per node depending on the total number of nodes for several vertex-connectivity parameters k for $k-$ regular k-connected graphs. In the worst cases, ($n = 100$ and $k = 34$), the data sent per node is around 500 KB, which is correct for nowadays technologies. Such a figure allows us to know the network cost, depending on the wanted robustness for several sizes of systems, up to 100 nodes.

On the others topologies NECTAR exhibits similar behaviors but seems less costly. In settings identical to those of Fig. 3, NECTAR is around 2 times less costly on $k-$diamond graphs and $k-$pasted graphs, and around 2.5 times less costly on multipartite wheel graphs and generalized wheel graphs.

Drone based scenario Fig. 4 shows the network cost per node in the drone scenario, depending on the distance (d) between the two barycenters, with $n = 20$ nodes. For $d = 0$ and $radius = 2.4$, the graph corresponds to a fully connected graph, and the data sent is around 50 KB. A distance $d = 6$ corresponds to a partitioned graph of two parts (it can be more for low values of $radius$). The red dotted curve corresponds to performances of MtG (that does not depend on d and $radius$): its amount of data sent is around 1.9 KB. Fig. 5 corresponds to the same experiment for MTGv2. In the worst cases, the amount of data sent is around 3 KB.

Number of nodes. The network cost per node in NECTAR increases in the worst cases quadratically with the number of nodes in the system and with the number of edges, as seen in Sec IV-E. To illustrate this effect, we now keep experimenting on regular and random graphs, but this time we vary the value of parameter n, for several values of d. The scope communication is fixed to $radius = 1.2$. Fig. 6 shows the results for NECTAR. The maximum value observed is around 200 KB per node, for $n = 50$, with $d = 0$, corresponding to almost fully connected graphs. Fig. 7 shows the results for MTGv2. The maximum amount of data sent per node observed is around 7.5 KB, for $n = 50$ and $d = 0$, also corresponding to almost fully connected graphs. Those values, both for NECTAR and MTGv2 algorithms, are very reasonable for nowadays technologies.

D. Byzantine resilience

Drone based scenario. This section investigates the resilience of NECTAR, MtG, and MtGv2 to Byzantine behaviors. We particularly study the robustness of MtG and MtGv2 protocols against few Byzantine nodes. Due to the use of Bloom filters, MtG is easily corruptible by Byzantine nodes.

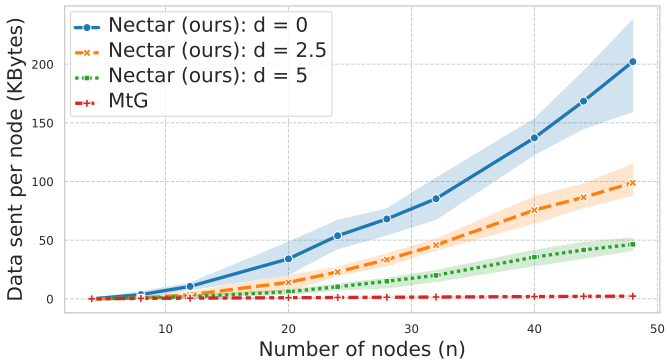

Fig. 6. data sent per node (in KB) depending on the number of nodes (n), for different values of distances (d) between barycenters, with a fixed communication scope (radius = 1.2), in the drone scenario.

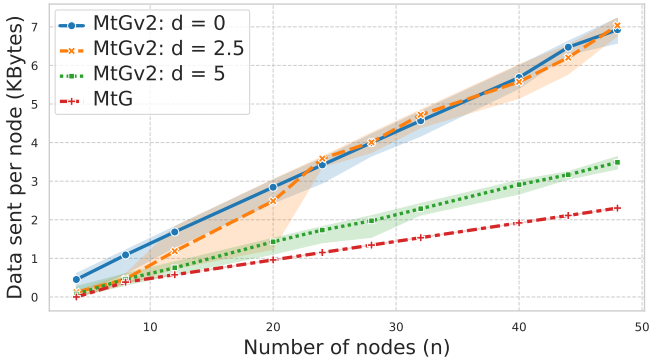

Fig. 7. data sent per node (in KB) depending on the number of nodes (n), for different values of distances (d) between barycenters, with a fixed communication scope (radius = 1.2), in in the drone scenario. The red line is the same as in Fig. 6.

For example, in a partitioned graph, Byzantine nodes can send filters full of 1 values to lead correct nodes to conclude that the system is connected. We experiment the situation of a graph partitioned into two parts, and we take care of equally distributing the Byzantine nodes between the two parts. The red dotted curve of Fig. 8 shows the proportion of correct nodes that correctly detect the partition. Such an experiment shows that two Byzantine nodes are enough to make all correct nodes reach the incorrect decision, while one Byzantine node is enough to prevent correct nodes from reaching the same decision (i.e., breaking the agreement property).

This attack is not possible on NECTAR and MtGv2, due to the use of signatures (Byzantine nodes cannot forge signatures). Thus, we considered another attack, which is the same for the two tested algorithms. We generated a subgraph of correct nodes that is partitioned into two parts. We then added Byzantine edges between each part, to make the graph connected, where all communications between the two correct parts must pass through Byzantine nodes, which also means that the graph is at most t-connected, where t is the number of Byzantines nodes, and the Byzantine nodes are the t key nodes that decide the connectivity parameter. The Byzantine behavior we considered for this kind of situation is that

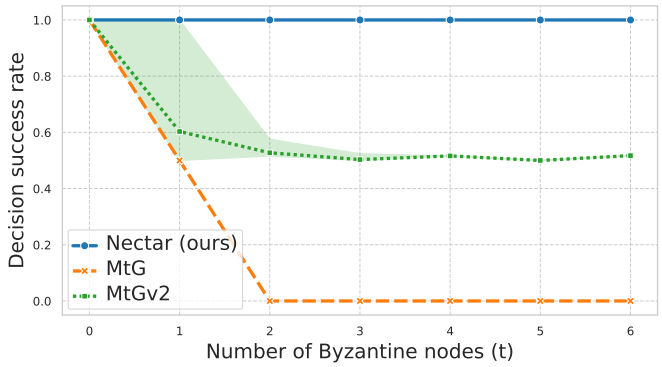

Fig. 8. Success rate of correct decision (through correct nodes), depending on the number of Byzantine nodes, for the drone scenario with 35 nodes. Results with 20 and 50 nodes exhibit the same tendencies.

Byzantine nodes act correctly toward one part of the subgraph of correct nodes, and as crashed nodes for the other part. Fig. 8 shows our results for these experiments. For NECTAR, as the connectivity will never be above t, all correct nodes conclude to a PARTITIONABLE decision, which is the correct decision since the subgraph of correct nodes is disconnected. For MtGv2, such an attack makes around half of the correct nodes conclude that the graph is connected (which is true), while the other half conclude that the graph is partitioned. Once again, one Byzantine node is enough to lead correct nodes to different decisions.

Connectivity-dependent topologies. We investigated the same attacks on the topologies highlighted by Bonomi et al. [23], with an aleatory placement of Byzantine nodes, and observed the following behaviors. For all topologies, MtG drops to 0 success rate of correct decision as soon as there are 2 Byzantine nodes, while NECTAR keeps a success rate of 1. For MtGv2, results depend on topologies: For $k-$diamond graphs, MtGv2 keeps a success rate close to 1 (with a confidence interval of [0.95, 1], no matter the number of Byzantine nodes. For $k-$regular graphs, $k-$pasted graphs, Generalised Wheel graphs, and Multipartite Wheel graphs, MtGv2 drops to 0.3 success rate on average, with a confidence interval of [0, 1].

To conclude, this experimental evaluation shows that existing partition detection algorithms are easily corruptible by a few Byzantine nodes using simple attacks such as sharing incorrect information or omitting to send some messages, although strengthening solutions by using signatures complicates attacks. NECTAR, however, provides a solution that ensures Byzantine resilience, no matter the network's topology. Although this solution is more costly than state-of-the-art ones, this cost is lower than 500 KB per node per algorithm execution, up to 100 nodes, which is a reasonable network cost for most of nowadays networks.

VI. RELATED WORK

A. Partition detection

Several works have studied how to detect network partition in fault-free and in crash-prone networks, in particular in

mobile ad hoc networks (MANETs) and Wireless Sensor Networks (WSNs). However, to the best of our knowledge, only [17] considered Byzantine faults.

1) In fault-free networks: Ritter et al. [4] proposed a heuristic to detect partitions in reliable MANETs. Their approach is based on the hypothesis that nodes at the border of the graph do not change frequently and are reliably detectable. Their idea is to exchange beacon messages through the network. A partition is suspected when beacons are not received for sufficiently long between well-chosen nodes.

Bouget et al. [6] propose MindTheGap (MtG), a light and fully decentralized approach dedicated to MANETs. MtG assumes that all nodes are correct but tolerates unreliable communication channels and detects partitions in dynamic networks. Every node in MtG gossips what it knows of the system using Bloom filters. Simulations revealed that MtG detects 90% of partitions despite a 40% message loss rate.

2) In crash-tolerant networks: Renesse et al. [7] propose a crash failure detection service based on gossip that can be used to detect partitions in asynchronous networks. Due to asynchrony, it is hard to know if a process (a node) has crashed or is just very slow. This is why, in such a context, allowing some false positive detections (assuming a slow process can be crashed) is reasonable while respecting acceptable accuracy. This failure detection algorithm is based on probabilistic properties and comes with guarantees about low rates of false detections, message-loss resilience, known probability of mistakes, and scalability. The authors turn their failure detection algorithm into a partition detection algorithm by detecting a group of unreachable nodes.

Conan et al. [8] propose an approach of partition detection based on heartbeat vector and propagation of reachability information for asynchronous distributed systems. Reachability information is obtained by propagating and aggregating path information for every node. Their method tolerates node mobility by frequently rebuilding the topology of the reachable system. An effort is made to distinguish disconnections from crashes. It is why, while detecting an unreachable node, computations are made to evaluate if the node is potentially just unreachable or crashed by paying attention to previous cases of potential disconnection of this node.

Those partition detection approaches have been designed for correct and crash-tolerant networks, and would not perform adequately under the Byzantine fault model that we consider.

Augustine et al. [17] address the problem of detecting if a graph is connected, in the context of Byzantine networks. While focusing on the very specific case of congested cliques, they provide an algorithm that detects if the subgraph of correct nodes is connected or if it is *far from connected*, i.e., if it has at least $2t+1$ connected component. The algorithm is executed in $O(polylog(n))$ time.

B. Reliable communication primitives

Dolev investigated the question of reliable communication in partially connected networks in the presence of Byzantine nodes [11]. He proved that agreement is possible if and only if $t < k/2$ and $t < n/3$ where t is the number of faulty nodes, k is the connectivity of the graph, and n is the number of nodes in the graph. While the $1/3$ of Byzantine nodes limit was well-known for fully connected networks [9, 10], this work identified the partial connectivity requirement for the first time. Dolev also introduced an algorithm that provides reliable communication, for $(2t+1)$-connected networks. The main idea of this algorithm is that traveling messages contain information about the path they followed in the network. By looking for that information, the nodes are able to compute the number of disjoint paths a message has traveled through. Nodes deliver a message when they are able to deduce that the message has traveled through $(t+1)$ disjoint paths. Two variants of this protocol are proposed, to deal with both known and unknown topologies. The message complexity of this protocol is high and, in the worst case, equal to $\mathcal{O}(n!)$ where n is the number of nodes.

This reliable communication protocol combined with Bracha's reliable broadcast algorithm [10] (introduced for fully connected networks) provides a reliable broadcast protocol for partially connected networks. However, since those two protocols are costly, some optimizations have been proposed in the state-of-the-art. Bonomi et al. [12] optimized the combination of Bracha's and Dolev's protocols through protocol-specific and cross-layer optimizations. Simulations showed that such optimizations made the protocols more practical by reducing its latency and the amount of transmitted information.

A new broadcast variant has been proposed by Khan et al. [26] that suits well partially connected networks: *local broadcast*. In the local broadcast context, nodes cannot send different values to their neighbors, even faulty nodes. This context makes sense in wireless communication, among others. Such a new primitive reduces the possible faulty behaviors of Byzantine nodes. Therefore, when local broadcast is possible, Khan et al. have shown that consensus primitives built on top of a local broadcast primitive have less restrictive limits than those built on point-to-point channels. Those limits are: $\lfloor 3t/2 \rfloor < k$ and $2t < d$, where k is the graph connectivity, t is the number of Byzantine nodes in the system, and d is the minimum node degree. Note that the constraint about graph connectivity is less restrictive. On the other hand, the restricted number of acceptable Byzantine nodes has been replaced by a condition about nodes degree. Recent works provided real-time guarantees for reliable broadcast in fully-connected networks with probabilistic message losses [27, 28]. We consider it future work to extend those guarantees to the partially connected networks we consider in this paper.

C. Vertex-connectivity in distributed systems

DECK is an algorithm that computes a network's connectivity [29]. It is fully decentralized and designed for asynchronous WSNets. However, it assumes that all nodes are correct when we consider Byzantine faults. Tucci Piergiovanni and Baldoni [30] proposed an adaption of the definition of vertex-connectivity for eventually quiescent dynamic dis-

tributed systems. Their work focuses, in particular, on the definition of strong connectivity in such systems, for which they propose an algorithm that computes a tree as an overlay topology that guarantees eventual connectivity.

D. Unknown systems and Detectors

Distributed unknown systems assume that nodes know neither the number of nodes in the systems nor their identity. Cavin et al. [31] first proposed a consensus algorithm for this type of systems based on a *Participant detector* oracle. They however suppose a reliable (and asynchronous) network. In unknown networks, Greve et al. [32] showed that consensus can be provided only if a node can establish (with the help of a *participant detector* oracle) that the network has a connectivity of at least $t+1$, where t is the maximal number of crashes. In Byzantine networks, Alchieri et al. [33] showed that Byzantine consensus can be achieved in unknown networks under the assumptions of synchronous communication, with sufficient connectivity in the network (connectivity of at least $2t+1$, where t is the number of faulty nodes). In these last works, both crash and Byzantine cases rely on the *participant detector* oracles (or variants) defined by Cavin et al [31].

In dynamic graphs, several works [34–36] have proposed a Byzantine failure detector. Byzantine failure detectors can be seen as oracles that detect when a Byzantine node deviates from its correct behavior. For our problem, we consider that Byzantine detectors are out of the scope since Byzantine nodes can attack with a behavior undifferentiable than correct nodes.

VII. Conclusion

In this paper, we investigated the question of partition detection in Byzantine networks. We highlight the necessity to reconsider the notion of partition for those cases of networks and propose a new *t-Byzantine-partitionable* notion. We characterize *t-Byzantine-partitionable* networks, relate this notion to the graph vertex-connectivity property, and propose a new algorithm NECTAR that detects such network property. NECTAR assumes synchronous communication and relies on signatures. We speculate that the problem becomes impossible to solve in an asynchronous environment, as in this case arbitrarily late messages become indistinguishable from the effect of a partition. Nevertheless, we posit that it can be accomplished without signatures in synchronous networks, albeit at a significant cost. We formally prove NECTAR correctness and propose an in-depth experimental evaluation. Our experimental evaluation aims to compare NECTAR to a non Byzantine-tolerant performant baseline and its signature-based variant, for several realistic families of topologies. Our results show that the NECTAR algorithm maintains a 100% accuracy of the tested scenarios while the accuracy of the best competitor baselines decreases by at least 40% as soon as one participant is Byzantine. Detecting t-Byzantine-partitionability in networks implies a higher network cost ($O(n^4)$ message complexity in the worst cases, i.e., in fully connected graphs) than efficient state-of-the-art partition detection algorithms. NECTAR's network cost increases with the number of nodes and decreases with the diameter of the networks, but it always remains lower than around 500 KB per node for up to 100 nodes.

Acknowledgments

This work was partially supported by the French ANR project ByBloS (ANR-20-CE25-0002-01) devoted to the modular design of building blocks for large-scale Byzantine-tolerant applications, by the EU Horizon Europe Research and Innovation Programme under Grant No. 101073920 (TENSOR) and by the Ecole Normale Supérieure (ENS) of Rennes.

References

[1] E. A. Brewer. "A certain freedom: thoughts on the CAP theorem". In: *PODC*. ACM, 2010.

[2] P. Ruiz and P. Bouvry. "Survey on broadcast algorithms for mobile ad hoc networks". In: *ACM Comp. Surveys (CSUR)* 48.1 (2015).

[3] K. Akkaya and M. Younis. "A survey on routing protocols for wireless sensor networks". In: *Ad hoc networks* 3.3 (2005).

[4] H. Ritter, R. Winter, and J. Schiller. "A partition detection system for mobile ad-hoc networks". In: *SECON*. IEEE, 2004.

[5] N. Martin. "Network partitioning algorithms with scale-free objective". PhD thesis. Feb. 2020.

[6] S. Bouget, Y.-D. Bromberg, H. Mercier, E. Riviere, and F. Taiani. "Mind the Gap: Autonomous detection of partitioned MANET systems using opportunistic aggregation". In: *SRDS*. IEEE, 2018.

[7] R. v. Renesse, Y. Minsky, and M. Hayden. "A gossip-style failure detection service". In: *Middleware*. ACM, 1998.

[8] D. Conan, P. Sens, L. Arantes, and M. Bouillaguet. "Failure, disconnection and partition detection in mobile environment". In: *NCA*. IEEE, 2008.

[9] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem". In: *ACM Trans. on Prog. Lang. and Sys.* 4.3 (1982).

[10] G. Bracha. "Asynchronous Byzantine agreement protocols". In: *Information and Computation* 75.2 (1987).

[11] D. Dolev. "Unanimity in an Unknown and Unreliable Environment". In: *FOCS*. IEEE, 1981.

[12] S. Bonomi, J. Decouchant, G. Farina, V. Rahli, and S. Tixeuil. "Practical Byzantine Reliable Broadcast on Partially Connected Networks". In: *ICDCS*. IEEE, 2021.

[13] M. Raynal. *Fault-Tolerant Message-Passing Distributed Systems*. Springer, 2018.

[14] R. Guerraoui, J. Komatovic, P. Kuznetsov, Y.-A. Pignolet, D.-A. Seredinschi, and A. Tonkikh. "Dynamic Byzantine Reliable Broadcast". In: *OPODIS*. 2021.

[15] A. Auvolat, D. Frey, M. Raynal, and F. Taïani. "Byzantine-Tolerant Causal Broadcast". In: *Theoretical Computer Science* 885 (2021).

[16] G. Zhang, F. Pan, M. Dang'ana, Y. Mao, S. Motepalli, S. Zhang, and H.-A. Jacobsen. *Reaching Consensus in the Byzantine Empire: A Comprehensive Review of BFT Consensus Algorithms*. 2022.

[17] J. Augustine, A. R. Molla, G. Pandurangan, and Y. Vasudev. "Byzantine Connectivity Testing in the Congested Clique". In: *Leibniz Int. Proc. in Informatics* (2022).

[18] D. Dolev and H. Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM J. Comput.* 12 (1983).

[19] B. N. Levine, C. Shields, and N. B. Margolin. "A survey of solutions to the sybil attack". In: *University of Massachusetts Amherst, Amherst, MA* (2006).

[20] K. Menger. "Zur allgemeinen kurventheorie". In: *Fundamenta Mathematicae* 10.1 (1927).

[21] D. B. Johnson and A. J. Menezes. "Elliptic curve DSA (ECDSA): an enhanced DSA". In: *USENIX Security*. 1998.

[22] D. Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux journal* 2014.239 (2014).

[23] S. Bonomi, G. Farina, and S. Tixeuil. "Multi-hop byzantine reliable broadcast with honest dealer made practical". In: *J. of the Brazilian Comp. Soc.* 25 (2019).

[24] A. Steger and N. C. Wormald. "Generating Random Regular Graphs Quickly". In: *Combinatorics, Probability and Computing* 8.4 (1999).

[25] R. Baldoni, S. Bonomi, L. Querzoni, and S. Tucci Piergiovanni. "Investigating the existence and the regularity of Logarithmic Harary Graphs". In: *Theoretical Computer Science* 410.21 (2009).

[26] M. S. Khan, S. S. Naqvi, and N. H. Vaidya. "Exact Byzantine Consensus on Undirected Graphs under Local Broadcast Model". In: *PODC*. ACM, 2019.

[27] D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo. "RT-ByzCast: Byzantine-resilient real-time reliable broadcast". In: *IEEE Transactions on Computers* 68.3 (2018), pp. 440–454.

[28] D. Kozhaya, J. Decouchant, V. Rahli, and P. Esteves-Verissimo. "Pistis: an event-triggered real-time byzantine-resilient protocol suite". In: *IEEE Transactions on Parallel and Distributed Systems* 32.9 (2021), pp. 2277–2290.

[29] V. K. Akram and O. Dagdeviren. "DECK: A distributed, asynchronous and exact k-connectivity detection algorithm for Wireless Sensor Networks". In: *Computer Communications* 116 (2018).

[30] S. Tucci Piergiovanni and R. Baldoni. "Connectivity in eventually quiescent dynamic distributed systems". In: *LADC*. 2007.

[31] D. Cavin, Y. Sasson, and A. Schiper. "Consensus with Unknown Participants or Fundamental Self-Organization". In: *ADHOC-NOW*. 2004.

[32] F. Greve and S. Tixeuil. "Knowledge Connectivity vs. Synchrony Requirements for Fault-Tolerant Agreement in Unknown Networks". In: *DSN'07*. 2007.

[33] E. A. P. Alchieri, A. Bessani, F. Greve, and J. d. S. Fraga. "Knowledge Connectivity Requirements for Solving Byzantine Consensus with Unknown Participants". In: *IEEE Trans. on Dependable and Secure Comp.* 15.2 (2018).

[34] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. "Byzantine Fault Detectors for Solving Consensus". In: *The Computer Journal* 46.1 (2003).

[35] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens. "An On-Demand Secure Routing Protocol Resilient to Byzantine Failures". In: *ACM Workshop on Wireless Security*. WiSE '02. New York, NY, USA: ACM, 2002.

[36] F. Greve, M. S. d. Lima, L. Arantes, and P. Sens. "A Time-Free Byzantine Failure Detector for Dynamic Networks". In: *EDCC*. 2012.