Knowledge Connectivity Requirements for Solving BFT Consensus with Unknown Participants and Fault Threshold

Hasan Heydari, Robin Vassantlal, and Alysson Bessani
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
{hheydari, rvassantlal, anbessani}@ciencias.ulisboa.pt

***Abstract*—Consensus is a fundamental building block for constructing reliable and fault-tolerant distributed services. The increasing demand for high-performance and scalable blockchain protocols has brought attention to solving consensus in scenarios where each participant joins the system knowing only a subset of participants. In such scenarios, the participants' initial knowledge about the existence of other participants can collectively be represented by a directed graph known as *knowledge connectivity graph*. The *Byzantine Fault Tolerant Consensus with Unknown Participants* (BFT-CUP) problem aims to solve consensus in those scenarios by identifying the necessary and sufficient conditions that the knowledge connectivity graphs must satisfy when a fault threshold is provided to all participants. This work extends BFT-CUP by eliminating the requirement to provide the fault threshold to the participants. We indeed address the problem of solving BFT consensus in settings where each participant initially knows a subset of participants, and although a fault threshold exists, no participant is provided with this information – referred to as *BFT Consensus with Unknown Participants and Fault Threshold* (BFT-CUPFT). With this aim, we first demonstrate that the conditions identified for knowledge connectivity graphs by BFT-CUP are insufficient to solve BFT-CUPFT. Accordingly, we introduce a new type of knowledge connectivity graph that is sufficient for solving such a problem. To validate its sufficiency, we design a protocol for solving BFT-CUPFT.**

***Index Terms*—Consensus with Unknown Participants, Byzantine Fault-Tolerance, Consensus, Blockchain.**

I. Introduction

Context. Consensus is a fundamental building block for constructing reliable and fault-tolerant distributed systems where participants agree on a common value out of the initially proposed values. This problem has been primarily addressed in three settings – permissioned, permissionless, and hybrid. In the permissioned setting, participants have a single global view of the system in advance, i.e., each participant is provided with the *system's membership* and the *fault threshold* (or, more generally, the fail-prone system [1]). Knowing such parameters simplifies the design and analysis of consensus protocols.

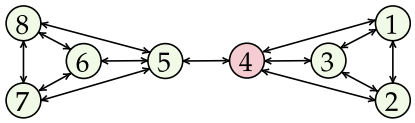
In the permissionless setting, protocols such as the Nakamoto consensus used in Bitcoin [2] solve consensus without requiring a single global view of the system. Specifically, no participant might be aware of the set of all participants. Furthermore, the fault threshold is not explicitly defined in the same way as in the permissioned setting; instead, these protocols rely on assumptions about the overall distribution of resources within their networks, like computational power in Bitcoin. Despite these protocols being scalable in terms of the number of participants, their performance is significantly lower by orders of magnitude compared to consensus protocols designed for the permissioned setting [3]–[5].

The demand to scale consensus to accommodate numerous participants while maintaining high performance has led to the emergence of the hybrid setting. Protocols tailored for this setting relax the global view requirements found in permissioned consensus protocols, allowing each participant to have a partial view of the participants that it can trust or initially knows. These protocols can be modeled in various ways. One approach is through the use of consensus with unknown participants (CUP) [6]–[11]. Alternatively, federated Byzantine quorum systems [12]–[15], personal Byzantine quorum systems [16], quorum systems designed for permissionless networks [17], and heterogeneous quorum systems [18] offer additional modeling options for hybrid consensus protocols. This paper focuses on the CUP model.

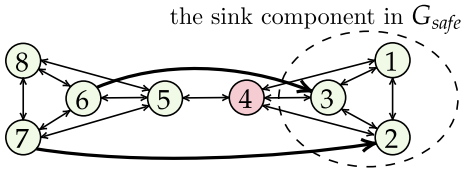
The CUP model. In scenarios where each participant joins the network initially knowing only a subset of participants, the knowledge about the existence of other participants can collectively be represented by a directed graph known as *knowledge connectivity graph* [6]. In such a graph, each vertex corresponds to a participant, and a directed edge from a vertex i to another vertex j denotes that participant i initially knows participant j. The graph depicted in Fig. 1a is an example of a knowledge connectivity graph in which participant 1 initially knows participants 2, 3, and 4.

It is crucial to emphasize that *the knowledge connectivity graph of a system might differ from its communication network*, as a participant i might be capable of communicating with another participant j; however, such communication cannot happen when i lacks knowledge about the existence of j (i.e., there is no link from i to j in their knowledge connectivity graph).

A knowledge connectivity graph must satisfy certain properties to allow participants to solve consensus. For instance, removing a faulty participant and its associated edges should not create multiple disconnected components, each having at least a correct participant. This is because if the faulty participant remains silent during the execution, the correct participants in each disconnected component may decide on a value independently of the correct participants in other components. This can result in multiple values being decided within the system, thereby failing to achieve consensus. For example, in the graph of Fig. 1a, in which only participant 4 is faulty,



(a) A knowledge connectivity graph that does not satisfy the requirements of the BFT-CUP model. Although the number of Byzantine participants is less than one-third of the total participants, satisfying the requirement for solving the traditional Byzantine consensus [19], solving consensus in this system is impossible when participant 4 remains silent, as participants in $\{1, 2, 3\}$ cannot acquire knowledge about participants in $\{5, 6, 7, 8\}$, and vice versa.



(b) A knowledge connectivity graph that satisfies the requirements of the BFT-CUP model; hence, participants can solve consensus even in the presence of a Byzantine participant.

Fig. 1: Two knowledge connectivity graphs. In each graph, a vertex corresponds to a participant, and the outgoing edges from a vertex i point to the participants that i initially knows; for instance, participant 1 initially knows participants 2, 3, and 4. Participant 4 is Byzantine, and others are correct.

correct participants cannot solve consensus if participant 4 remains silent.

The CUP model determines the *necessary* and *sufficient* properties that a knowledge connectivity graph must satisfy to allow solving consensus under specific synchrony and fault assumptions. In this model, each participant is provided with the system's fault threshold and a subset of other participants when joining the network. As synchrony and fault assumptions are relaxed, a participant's knowledge must be increased, i.e., the number of its outgoing edges in the knowledge connectivity graph must be increased. For example, each participant in the Byzantine Fault-Tolerant (BFT) CUP [10] requires more knowledge than in the fault-free CUP model [6]. The graph depicted in Fig. 1b illustrates an example of a knowledge connectivity graph that satisfies the requirements of the BFT-CUP model.

Problem statement. The BFT-CUP model solves consensus in partially synchronous systems, where each participant initially knows the system's fault threshold and a subset of participants. Since knowing the system's fault threshold restricts the full potential of the BFT-CUP model in the hybrid setting, the main goal of this paper is to eliminate the explicit knowledge of that parameter by proposing a new model called the *BFT Consensus with Unknown Participants and Fault Threshold* (BFT-CUPFT). The BFT-CUPFT model is indeed an extension of BFT-CUP that enables participants to solve BFT consensus in partially synchronous systems where each participant initially knows only a subset of participants.

Contributions. To achieve such a goal, we begin by revisiting the BFT-CUP model under the extra assumption that each participant can use digital signatures (this assumption has been made in related work [11], [13], [14]). We refer to the resulting model as the *authenticated BFT-CUP* model. Notably, Alchieri et al. [9] established that the requirements of knowledge connectivity graphs for solving consensus in the BFT-CUP model remain unchanged, irrespective of whether digital signatures are used. Although such an assumption does not reduce the initial knowledge required by participants to solve consensus, it enables the design of a simpler consensus protocol. We use the simpler protocol to highlight the significance of the fault threshold in the BFT-CUP model and clarify the necessary adjustments required when participants lack such information.

We then show an impossibility result stating that solving consensus is impossible in a partially synchronous system when the knowledge connectivity graph formed collectively by the initial knowledge of participants satisfies the requirements of the BFT-CUP model, yet no participant is provided with the fault threshold. This holds even when considering the aforementioned extra assumption. As a result, we introduce the *BFT Consensus with Unknown Participants and Fault Threshold* (BFT-CUPFT) model, defining a new type of knowledge connectivity graph that is sufficient to solve consensus without having information about the fault threshold. To validate its sufficiency, we design a protocol for solving consensus in BFT-CUPFT. Table I outlines the (im)possibility of solving BFT consensus deterministically under different system models.

In summary, the main contributions of this paper can be outlined as follows:

- We revisit the BFT-CUP model by assuming each participant can use digital signatures. We design protocols that are simpler than the original ones under this new assumption.
- We prove an impossibility result for solving consensus when knowledge connectivity graphs satisfy the requirements of the BFT-CUP model, but participants are not provided with the fault threshold.
- We introduce the BFT-CUPFT model by defining a new type of knowledge connectivity graph that allows participants to solve consensus when each participant initially knows only a subset of participants, not the system's membership or the fault threshold.
- We design a consensus protocol that solves consensus in the BFT-CUPFT model.

Paper organization. The remainder of the paper is organized as follows. Section II introduces our system model and provides the background for this paper. Section III revisits the BFT-CUP model by assuming that each process can use digital signatures. Section IV presents an impossibility result for solving consensus when knowledge connectivity graphs satisfy the requirements of the BFT-CUP model, but participants are not provided with the fault threshold. Section V introduces the BFT-CUPFT model, defining a new type of knowledge connectivity graph. Section VI solves consensus in the BFT-

		Knowledge about n and f		
		Known n, Known f	Unknwon n, Known f	Unknown n, Unknown f
Communication Model	Sync.	✓ (e.g. [20])	✓ (e.g. [21])	✓ (e.g. [21])
	Partially Sync.	✓ (e.g. [22], [23])	✓ (BFT-CUP [9], [10])	✓ (BFT-CUPFT) ← this work
	Async.	✗ (see [24])	✗ (see [24])	✗ (see [24])

TABLE I: The (im)possibility of solving Byzantine consensus deterministically under different system models.

CUPFT model. Sections VII and VIII present the related work and conclude the paper, respectively.

II. Preliminaries

A. System Model

We consider a distributed system comprised of a finite set Π of processes operating under the assumption of partial synchrony [19], [25], which guarantees that, for each execution of the protocol, there exists a time GST and a duration δ such that after GST, message delays between correct processes are bounded by δ. Before GST, messages may experience arbitrary delays. We assume that each process $i \in \Pi$ initially knows only a subset $\Pi_i \subseteq \Pi$.

We denote the set of failed processes during an execution by $\Pi_F \subset \Pi$. The faulty processes can behave arbitrarily, i.e., can be Byzantine [26], and may collude and coordinate their actions. A non-Byzantine process is said to be *correct*. We denote the set of correct processes during an execution by $\Pi_C = \Pi \setminus \Pi_F$. We assume a static Byzantine adversary controlling the faulty processes, i.e., the set of processes controlled by the adversary is fixed at the beginning and does not change throughout the execution of the protocols. We assume that correct processes neither know Π nor Π_F; however, Byzantine processes may know both.

We further assume that each process has a unique ID, IDs are not necessarily consecutive, and it is infeasible for a faulty process to obtain additional IDs to launch a *Sybil attack* [27]. Processes communicate by message passing through authenticated and reliable point-to-point channels, and when needed, can sign messages using digital signatures. We denote by $\langle m \rangle_i$ a message m signed by process i. A process i can only send a message directly to another process j if i knows j, i.e., if $j \in \Pi_i$.

B. The Consensus Problem

In the consensus problem, each process *proposes* a value, and all correct processes must *decide* the same value among the proposed values. Formally, any protocol that solves consensus must satisfy the following properties:

- *Validity:* if a correct process decides a value v, then v was proposed by some process.
- *Agreement:* no two correct processes decide differently.
- *Termination:* every correct process eventually decides some value.
- *Integrity:* every correct process decides at most once.

C. The BFT-CUP Model

The *Byzantine Fault-Tolerant Consensus with Unknown Participants* (BFT-CUP) model [10] solves consensus while tolerating Byzantine faults in partially synchronous systems. The BFT-CUP model operates under the same system model as this paper, except it does not rely on digital signatures. It also assumes every process is provided with the fault threshold f as input (i.e., $|\Pi_F| \leq f$).

In this model, each process initially obtains partial knowledge about the others using a local oracle called *participant detector* (PD). In further detail, let PD_i represent the participant detector for process i, wherein PD_i returns a set $\Pi_i \subseteq \Pi$ comprising the processes that i can initially contact. We assume that Π_i can expand over time, but PD_i always returns the same set. Further, we assume that PD_i might return Π if i is Byzantine as by assumption, a Byzantine process may know Π.

The information provided by the participant detectors of all processes collectively forms a directed graph known as *knowledge connectivity graph* [6]. Specifically, a directed graph $G_{di} = (V_{di}, E_{di})$ is a knowledge connectivity graph if V_{di} represents the set of processes Π, and an edge $(i, j) \in E_{di}$ exists if and only if process i initially knows process j, i.e., $j \in PD_i$. For instance, the graphs illustrated in Fig. 1 exemplify knowledge connectivity graphs wherein $PD_1 = \{2, 3, 4\}$. Since the PD of each process always returns the same set, the knowledge connectivity graph formed by PDs of processes is a *static* graph.

As stated before, the system's knowledge connectivity graph might differ from its communication network. In further detail, in the BFT-CUP model, the communication network is assumed to be a complete graph as there is a reliable point-to-point channel between any two processes, according to the system model. However, the knowledge connectivity graph is not necessarily a complete graph (e.g., see Fig. 1). When $j \in PD_i$, process i can send messages to j from the beginning of the execution. However, when $j \notin PD_i$, process i first needs to discover j through the processes it knows, and only after that it can directly communicate with j.

The BFT-CUP model guarantees that processes can solve consensus when the knowledge connectivity graph meets specific requirements. Before delving into these requirements, we first review some relevant graph notations. A directed graph $H_{di} = (V_{di}, E_{di})$ is said to be *k-strongly connected* if for any pair of nodes $i, j \in V_{di}$, i can reach j through at least k node-disjoint paths in H_{di}. The *strong connectivity* $\kappa(H_{di})$ of H_{di} is the maximum value of k for which H_{di} is k-strongly

connected. Given two non-empty sets of processes A and B, we use the notation $A \xrightarrow{k} B$ to indicate that there are k vertices in A that have outgoing neighbors in B, where k is a natural number. Besides, $A \xrightarrow{>k} B$ denotes that the number of vertices in A that have outgoing neighbors in B is greater than k.

Given an directed graph $G_{di} = (V_{di}, E_{di})$, we denote the subgraph induced by a set of nodes $U_{di} \subseteq V_{di}$ as $G_{di}[U_{di}]$, i.e., $G_{di}[U_{di}] = (U_{di}, \{(i,j) \mid i,j \in U_{di} \wedge (i,j) \in E_{di}\})$. Further, we say a strongly connected component G_{sink} of G_{di} is a *sink* if and only if there is no path from a node in G_{sink} to other nodes of G_{di}, except nodes in G_{sink} itself. A node $v \in V_{di}$ is a *sink member* if it belongs to a sink component of G_{di}; otherwise, it is a *non-sink member*. An undirected counterpart can be defined for any directed graph G_{di} as $G = (V_{di}, \{(i,j) \mid (i,j) \in E_{di} \vee (j,i) \in E_{di}\})$.

Definition 1 (k-One Sink Reducibility (k-OSR) PD [10])**.** A graph G_{di} belongs to k-OSR PD if:

- the undirected graph G obtained from G_{di} is connected,
- the directed acyclic graph obtained by reducing G_{di} to its strongly connected components has exactly one sink, namely $G_{sink} = (V_{sink}, E_{sink})$,
- the sink component G_{sink} is k-strongly connected, and
- there are at least k node-disjoint paths from any process $i \notin V_{sink}$ to any process $j \in V_{sink}$.

Given a knowledge connectivity graph $G_{di} = (V_{di}, E_{di})$, we refer to $G_{di}[\Pi_C]$ by the *safe* subgraph G_{safe} of G_{di}. With these definitions, we are ready to present the requirements that a knowledge connectivity graph must satisfy to enable solving BFT-CUP.

Theorem 1 ([10])**.** The safe subgraph G_{safe} of a knowledge connectivity graph must satisfy the following two properties to ensure correct processes can solve BFT-CUP:

- G_{safe} belongs to the $(f+1)$-OSR PD, and
- the sink component of G_{safe} must contain at least $2f+1$ processes.

If a knowledge connectivity graph meets the properties of Theorem 1, we say it satisfies the requirements of the BFT-CUP model. We also denote the family of all finite graphs that meet these properties by $\mathcal{G}_{di}$. Note that a graph that belongs to $\mathcal{G}_{di}$ can have up to f Byzantine nodes, but the properties specified in Theorem 1 only describe the characteristics of the subgraph induced by correct nodes. Fig. 1b is an example of a knowledge connectivity graph that satisfies the requirements of the BFT-CUP model in which participants in $\{1,2,3\}$ are the sink members of G_{safe}. Note that participant 4 is Byzantine and therefore not considered in G_{safe}.

Solving consensus in the BFT-CUP model. The sink component holds a pivotal role in the BFT-CUP model. In order to solve consensus within this model, processes expand their knowledge – i.e., the set of processes they initially know – and actively seek to identify the sink component. Processes communicate using a communication primitive called reachable reliable broadcast [10] that allows delivery of a message if it is received through more than f node-disjoint paths from the sender.

Once the sink component is discovered, processes can establish intersecting quorums and execute a consensus protocol, such as PBFT [22], on top of those quorums. The definition of quorums highlights the significance of the sink: as demonstrated in [11], any defined quorum must include at least $\lceil (|V_{sink}| + f + 1)/2 \rceil$ sink processes to intersect with any other quorum in at least one correct process.

Remark 1. The BFT-CUP model does not rely on digital signatures.

III. Revisiting the BFT-CUP Model

This section revisits the BFT-CUP model, incorporating an additional assumption that allows each process to use digital signatures – denoting the resulting model as the *authenticated BFT-CUP* model. Recall that in the BFT-CUP model, a process delivers a message if it is received through more than f node-disjoint paths. However, using digital signatures, processes can deliver messages without waiting to receive them through multiple paths. This significantly simplifies the communication protocols and reduces the places where the fault threshold is required. As presented next, the consensus protocol in the authenticated BFT-CUP model has roughly 20 lines compared with 120 in [10]. Although the requirements of the knowledge connectivity graphs to solve consensus remain unchanged [9], in the simplified protocol, we can highlight where the fault threshold is still required and clarify the necessary adjustments to remove it entirely.

A. Consensus Protocol in the Authenticated BFT-CUP Model

In order to solve consensus in the authenticated BFT-CUP model, we introduce three algorithms – namely Discovery, Sink, and Consensus – along with their associated properties. The relationship between these algorithms is as follows: the Consensus algorithm executes the Sink algorithm, and the Sink algorithm executes the Discovery algorithm.

Discovery algorithm. The Discovery algorithm, described in Algorithm 1, enables any correct process to expand the set of processes that it knows with the aim of eventually receiving the PD of each correct process reachable from it. This algorithm provides only one task, `discovery`. When a process executes this task, it periodically asks the processes it knows to respond by sending the PDs they have received. Each process i has the following three local sets:

- $\mathcal{S}^i_{PD}$ – Process i stores any received PD in this set, initialized with $\{\langle i, PD_i \rangle_i\}$.
- $\mathcal{S}^i_{known}$ – This set contains the processes that i knows, initialized with $PD_i \cup \{i\}$.
- $\mathcal{S}^i_{received}$ – This set contains the set of processes that i has received their PDs, initialized with $\{i\}$.

Periodically, each process i sends a $\langle$GETPDS$\rangle$ message to all processes that it knows, requesting each of them to share its $\mathcal{S}^*_{PD}$ (line 2). Upon receiving a $\langle$GETPDS$\rangle$ request from process j, process i responds by sending the PDs it knows,

Algorithm 1 The Discovery algorithm – process i.

task `discovery()`
1: $\mathcal{S}^i_{PD}, \mathcal{S}^i_{known}, \mathcal{S}^i_{received} \leftarrow \{\langle i, PD_i\rangle_i\}, PD_i \cup \{i\}, \{i\}$
2: **periodically** $\forall j \in \mathcal{S}^i_{known}$: **send** $\langle$GETPDS$\rangle$ **to** j

upon receiving $\langle$GETPDS$\rangle$ **from** j
3: **send** $\langle$SETPDS$, \mathcal{S}^i_{PD}\rangle$ **to** j

upon receiving $\langle$SETPDS$, \mathcal{S}^j_{PD}\rangle$ **from** j
4: $\mathcal{S}^i_{PD} \leftarrow \mathcal{S}^i_{PD} \cup \mathcal{S}^j_{PD}$
5: $\mathcal{S}^i_{known} \leftarrow \mathcal{S}^i_{known} \cup \{k \in PD_* \mid \langle *, PD_*\rangle_* \in \mathcal{S}^j_{PD}\}$
6: $\mathcal{S}^i_{received} \leftarrow \mathcal{S}^i_{received} \cup \{k \mid \langle k, *\rangle_* \in \mathcal{S}^j_{PD}\}$

i.e., sending set $\mathcal{S}^i_{PD}$ to j (line 3). When i receives a message $\langle$SETPDS$, \mathcal{S}^j_{PD}\rangle$ from process j, it updates its local sets using $\mathcal{S}^j_{PD}$ (lines 4-6). It is worth noting that, since correct processes sign their PDs (line 1), Byzantine processes cannot lie about the PD of any correct process i, either by modifying PD_i or by creating a PD for i.

This algorithm satisfies two properties that are specified in the following theorem.

Theorem 2. Consider a system with a knowledge connectivity graph $G_{di} \in \mathcal{G}_{di}$. Assuming V_{sink} comprises the sink members of G_{di}, by executing Algorithm 1 in this system, every correct process eventually (a) discovers all correct sink members, i.e., $\forall i \in \Pi_C : V_{sink} \cap \Pi_C \subseteq \mathcal{S}^i_{known}$, and (b) receives the PDs of all correct sink members, i.e., $\forall i \in \Pi_C : V_{sink} \cap \Pi_C \subseteq \mathcal{S}^i_{received}$.

The proof of the aforementioned theorem, along with proofs of other theorems not presented here, can be found in the extended version of this paper [28].

Sink algorithm. The Sink algorithm enables each process to discover a sink component. Before introducing this algorithm, we present two theorems for a knowledge connectivity graph belonging to $\mathcal{G}_{di}$. The first theorem ensures the existence of two sets with specific properties, while the second theorem demonstrates that the union of these sets contains all and only sink members.

Theorem 3. In a knowledge connectivity graph $G_{di} = (V_{di}, E_{di})$ belonging to $\mathcal{G}_{di}$, there are two sets $S_1, S_2 \subseteq V_{di}$ that satisfy the following properties for a given fault threshold f:

P1) The size of S_1 is greater than or equal to $2f + 1$, i.e., $|S_1| \geq 2f + 1$.
P2) The strong connectivity of the subgraph induced by S_1 is greater than or equal to $f + 1$, i.e., $\kappa(G_{di}[S_1]) \geq f + 1$.
P3) The number of processes in S_1 that have outgoing edges to $V_{di} \setminus S_1$ is at most f, i.e., $S_1 \xrightarrow{\leq f} V_{di} \setminus S_1$.
P4) S_2 contains any process $i \notin S_1$ such that the number of processes in S_1 that have outgoing edges to i is greater than f, i.e., $\forall i \in V_{di} \setminus S_1 : S_1 \xrightarrow{>f} \{i\} \iff i \in S_2$.

Theorem 4. In a knowledge connectivity graph $G_{di} = (V_{di}, E_{di})$ belonging to $\mathcal{G}_{di}$, if a fault threshold f and two subsets of processes S_1 and S_2 satisfy the properties specified in Theorem 3, then set $S_1 \cup S_2$ contains all and only the sink members.

In Theorems 3 and 4, set S_1 (resp. S_2) comprises the subset of sink members whose strong connectivity can (resp. cannot) be calculated. To clarify why the strong connectivity of S_2 cannot be calculated, we determine sets S_1 and S_2 in two scenarios. In these scenarios, we assume that each sink member executes the Discovery algorithm. For simplicity, we assume there are f Byzantine processes in the sink. This happens when each correct sink member has at least $f + 1$ node-disjoint paths to each Byzantine sink member, i.e., for each Byzantine sink member i, there are at least $f + 1$ correct sink members that know i.

- **Scenario I)** Byzantine sink members remain silent during the execution, and every correct sink member receives the PDs of other correct sink members by time $t \geq 0$. At time t, S_1 contains all correct sink members while S_2 contains all Byzantine sink members. Note that each correct sink member can calculate the strong connectivity of S_1 at time t, while it cannot calculate the strong connectivity of S_2 or any other set containing at least a Byzantine sink member. This limitation arises because for computing the strong connectivity of a set, it is required to have the PDs of all processes in that set.
- **Scenario II)** By time t, each sink member receives the PDs of other sink members, except for f correct sink members D. Since a correct process that is not a member of D cannot distinguish between this and the previous scenario, it determines sets S_1 and S_2 at time t, considering the possibility that members of D might be Byzantine and stay silent. Accordingly, S_1 contains those sink members that are not in set D while $S_2 = D$. Like the previous case, as members of S_1 have not received the PDs of members of S_2, they cannot calculate the strong connectivity of S_2 or any set that contains at least a member of D.

The following steps provide an insight into why the properties presented in Theorem 4 identify a sink:

- Since $|\Pi_F| \leq f$, Property P1 implies that S_1 contains at least $f + 1$ correct processes.
- Since $G_{di} \in \mathcal{G}_{di}$, there is no link from a correct sink member to a correct non-sink member. Hence, S_1 cannot contain both correct sink and correct non-sink members due to Property P2. Accordingly, all correct members of S_1 are either sink members or non-sink members.
- For the sake of contradiction, assume that all correct members of S_1 are non-sink members. Since $G_{di} \in \mathcal{G}_{di}$, any correct non-sink member has at least $f + 1$ node-disjoint paths to correct sink members. Therefore, all correct members of S_1 must have at least $f + 1$ node-disjoint paths to the sink members. This implies that there are at least $f + 1$ outgoing edges from S_1 to the remaining processes. Consequently, S_1 cannot satisfy

Algorithm 2 The Sink algorithm – process i.

function $\texttt{isSink}_{G_{di}}(f, S_1, S_2)$

1: **return** $|S_1| \geq 2f+1 \wedge \kappa(S_1) \geq f+1 \wedge (S_1 \xrightarrow{\leq f} \mathcal{S}^i_{known} \setminus S_1) \wedge S_2 = \{j \mid \forall j \in \mathcal{S}^i_{known} \setminus S_1 : S_1 \xrightarrow{>f} \{j\}\}$

function $\texttt{sink}()$

2: **fork** $\texttt{discovery}()$

3: **wait until** $\exists S_1 \subseteq \mathcal{S}^i_{received}, \exists S_2 \subseteq \mathcal{S}^i_{known} \setminus S_1 :$ $\texttt{isSink}_{G_{di}}(k-1, S_1, S_2) = true$

4: **return** $S_1 \cup S_2$

Property P3, which means all correct members of S_1 are sink members.

- For the sake of contradiction, assume that $S_1 \cup S_2$ is a proper subset of a sink, i.e., there is at least a process $i \in V_{sink}$ such that $i \notin S_1 \cup S_2$. Since $G_{di} \in \mathcal{G}_{di}$, there are at least $f+1$ node-disjoint paths from members of S_1 to i. Accordingly, i must be a member of S_2, which is a contradiction. Hence, $S_1 \cup S_2$ contains all members of a sink.

We define a predicate $\texttt{isSink}_{G_{di}}$ to check whether a given fault threshold f and two sets of processes S_1 and S_2 satisfy the properties specified in Theorem 3. Specifically, $\texttt{isSink}_{G_{di}}(f, S_1, S_2)$ is *true* if and only if these parameters meet the properties of such a theorem.

Taking into account the existence of Byzantine processes in the sink, the Sink algorithm (Algorithm 2) enables each process i to discover the sink members when the knowledge connectivity graph $G_{di} \in \mathcal{G}_{di}$. This algorithm provides only one function, `sink`. Using this function, each correct process continuously expands the set of processes it knows by executing the Discovery algorithm until it identifies the sink component. It terminates the Sink algorithm by returning a set equal to V_{sink} when there exists two sets S_1 and S_2 with conditions specified in line 3. These conditions are analogous to the properties specified in Theorem 3.

Using an example, we demonstrate why the Sink algorithm returns $S_1 \cup S_2$ as the sink. Consider the knowledge connectivity graph depicted in Fig. 1b. Suppose process 1 executes the `sink` function, and consider the following scenario. Process 2 is slow, and process 4 sends a set $P = \{1, 2, 3\}$ as its PD during the execution of `discovery`. When process 1 receives P and PD_3, the conditions specified in line 3 of Algorithm 2 are satisfied with $S_1 = \{1, 3, 4\}$. Note that there are more than f processes in S_1 that have outgoing edges to process 2 so $S_2 = \{2\}$. Consequently, set $S_1 \cup S_2 = \{1, 2, 3, 4\}$ is returned as the sink. It is worth noting that in this algorithm, the connectivity of S_1 can be computed as $S_1 \subseteq \mathcal{S}^*_{received}$, while the connectivity of S_2 cannot be computed since the PDs of processes in S_2 were not received.

The Sink algorithm satisfies the properties specified in the following theorem.

Theorem 5. In a knowledge connectivity graph $G_{di} \in \mathcal{G}_{di}$, Algorithm 2 executed by any correct process (a) eventually terminates and (b) returns all sink members.

Algorithm 3 The Consensus algorithm – process i.

variables

1: $val \leftarrow \perp$

function $\texttt{propose}(v)$

2: $S \leftarrow \texttt{sink}()$

3: **if** $i \in S$

4: $\quad val \leftarrow \texttt{consensus.propose}(S, v)$

5: **else**

6: $\quad \forall j \in S$: **send** $\langle$GETDECIDEDVAL$\rangle$ **to** j

7: $\quad$ **wait until receiving the same** $\langle$DECIDEDVAL, $val\rangle$ **from** $\lceil(|S|+1)/2\rceil$ distinct processes in S

8: **return** val

upon receiving $\langle$GETDECIDEDVAL$\rangle$ **from** j

9: **wait until** $val \neq \perp$

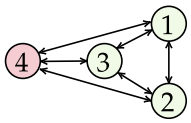
10: **send** $\langle$DECIDEDVAL, $val\rangle$ **to** j

Solving consensus in the authenticated BFT-CUP model. Algorithm 3 solves consensus in the authenticated BFT-CUP model. This algorithm provides a `propose` function through which processes can propose a value and decide on a common value. In this algorithm, each process i first executes the Sink algorithm. Upon the termination of the Sink algorithm, process i acts based on whether it is a sink or a non-sink member. If it is a sink member, it executes a traditional consensus protocol (e.g., PBFT [22]) with the sink members. Otherwise, it asks the sink members to respond by sending the decided value and waits until receiving the same value v from $\lceil(|S|+1)/2\rceil$ distinct sink members, where S comprises the sink members. Since the sink component contains at least $2f+1$ correct and at most f Byzantine processes, $\lceil(|S|+1)/2\rceil \geq f+1$. This implies that there is at least one correct process among the answering processes. Thus, a correct process can be sure that v is the value decided by correct sink members and, therefore, can decide v.

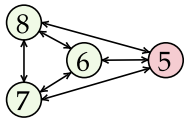
Theorem 6. Algorithm 3 solves consensus in the authenticated BFT-CUP model.

B. The Fault Threshold's Role in the BFT-CUP Model

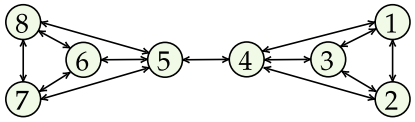
In the three algorithms presented to solve consensus in the authenticated BFT-CUP model, the only place where the fault threshold is used is in the Sink algorithm (Algorithm 2). In that algorithm, each process requires knowing f in order to identify the sink and terminate the algorithm. Consequently, the lack of access to f may lead to the following issues: (a) the Sink algorithm does not terminate, (b) multiple subsets of the sink might declare themselves as the sink, or (c) a subset of non-sink members might declare themselves as the sink. As processes must discover the sink to solve consensus, the Termination property of consensus is violated if the first scenario happens. If the second and third scenarios happen, the Agreement property of consensus can be violated as members



(a) System A: a 2-OSR PD in which only process 4 is faulty.



(b) System B: a 2-OSR PD in which only process 5 is faulty.



(c) System AB: a 1-OSR PD in which all processes are correct.

Fig. 2: Processes in $\{1, 2, 3\}$ cannot distinguish between case a (if process 4 remains silent) and case c (if process 4 is slow). Likewise, Processes in $\{6, 7, 8\}$ cannot distinguish cases b and c.

of each sink can solve consensus independently of other processes.

IV. An Impossibility Result

The primary objective of this paper is to extend the BFT-CUP model to enable solving consensus when each process only knows its PD but does not know f. We achieve this by finding the sufficient knowledge connectivity requirements for solving consensus in this setting.

The initial step for finding such requirements involves addressing the following question: *in partially synchronous systems, is having a knowledge connectivity graph that satisfies the requirements of the BFT-CUP model sufficient to solve consensus when the fault threshold is unknown?* That is, in a partially synchronous system, can processes solve consensus when each process i initially has access to its participant detector PD_i, the sets returned by participant detectors of processes collectively form a knowledge connectivity graph $G_{di} \in \mathcal{G}_{di}$, and no correct process knows the value of f? Note that the system still has a fault threshold as required to define G_{di}, but it is unknown. We negatively answer the above question by presenting a theorem that relies on the indistinguishability technique (e.g., used in [24], [29], [30]).

Theorem 7. In partially synchronous systems, a knowledge connectivity graph belonging to $\mathcal{G}_{di}$ is insufficient to solve Byzantine consensus when the fault threshold is unknown.

Proof. We establish the theorem for a weaker failure model, specifically crash faults, by assuming that a process fails by crashing. Given that any impossibility result derived for a weaker model holds for a stronger model, an impossibility result drawn for crash faults is also valid for Byzantine faults.

For the sake of contradiction, assume that there is a protocol $\mathcal{A}$ by which processes can solve consensus when the knowledge connectivity graph of the system belongs to $\mathcal{G}_{di}$, but no process knows the value of the fault threshold. We present three cases, each with a corresponding knowledge connectivity graph. It is straightforward to validate that each knowledge connectivity graph belongs to $\mathcal{G}_{di}$.

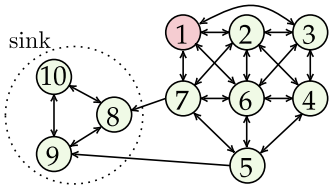
a) Consider a distributed system A composed of a set of processes $\Pi_A = \{1, 2, 3, 4\}$ with the knowledge connectivity graph depicted in Fig. 2a. Except for process 4, other processes are correct. Assume that the initial value of every process is v. Additionally, assume that the GST occurs at most by time t_A in this system (in accordance with the definition of partial synchrony, making such an assumption is possible). By assumption, processes $1, 2$, and 3 must be able to solve consensus using $\mathcal{A}$. Let E_A be an execution of $\mathcal{A}$ with duration Δ_A, resulting in deciding v due to the Validity property of consensus.

b) Similar to the previous case, consider a distributed system B composed of a set of processes $\Pi_B = \{5, 6, 7, 8\}$ with the knowledge connectivity graph depicted in Fig. 2b. Except for process 5, other processes are correct. Assume that the initial value of every process is u. Additionally, assume that the GST occurs at most by time t_B in this system. By assumption, processes $6, 7$, and 8 must be able to solve consensus using $\mathcal{A}$. Let E_B be an execution of $\mathcal{A}$ with duration Δ_B, resulting in deciding u due to the Validity property of consensus.

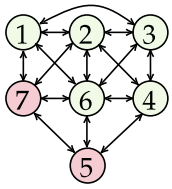
c) Consider a distributed system AB composed of eight processes $\Pi_A \cup \Pi_B = \{1, 2, \ldots, 8\}$ with the knowledge connectivity graph depicted in Fig. 2c. Assume that all processes are correct and the initial value of each member of Π_A (resp. Π_B) is v (resp. u). Furthermore, assume that the communication delays between any two members of $\{1, 2, 3\}$ (resp. $\{6, 7, 8\}$) are the same as the communication delays in system A (resp. B). However, any message sent between any other two processes will be received after $\texttt{max}\{t_A + \Delta_A, t_B + \Delta_B\}$. Note that processes $1, 2$, and 3 cannot distinguish cases a and c, so they must decide v. Likewise, processes $6, 7$, and 8 cannot distinguish cases b and c, so they must decide u. Thus, the Agreement property is violated.

The violation of the Agreement property in the third case implies a contradiction. Consequently, our assumption that states there is a protocol by which processes can solve consensus when the fault threshold is unknown in a system with a knowledge connectivity graph belonging to $\mathcal{G}_{di}$ is incorrect, completing the proof. □

The above theorem describes executions where, before deciding a value, no process in $\{1, 2, 3\}$ can discover a process in $\{6, 7, 8\}$ and vice versa. Note that in system AB, processes in $\{1, 2, 3\}$ (resp. $\{6, 7, 8\}$) can consider themselves as sink members since they are sink members in system A (resp. system B), and they cannot make a distinction between systems A and AB (resp. systems B and AB). That is when processes do not know f, since $\texttt{isSink}_{G_{di}}(1, \{1, 2, 3\}, \{4\}) = \mathit{true}$, and $\texttt{isSink}_{G_{di}}(1, \{6, 7, 8\}, \{5\}) = \mathit{true}$, sets $\{1, 2, 3, 4\}$ and $\{5, 6, 7, 8\}$ are sinks. This result gives rise to the following observation:



(a) System A: a 2-OSR PD in which only process 1 is faulty.



(b) System B: a 3-OSR PD in which processes 5 and 7 are faulty.

Fig. 3: Processes in $\{2, 3, 4, 6\}$ cannot distinguish between case a (if processes 5 and 7 remain silent) and case b (if process 1 behaves like correct processes but processes 5 and 7 are slow).

Observation 1. In partially synchronous systems, when the fault threshold is unknown, and the knowledge connectivity graph belongs to $\mathcal{G}_{di}$, if there exists a natural number g and two subsets of processes S_1 and S_2 such that $\texttt{isSink}_{G_{di}}(g, S_1, S_2) = true$, then processes in S_1 consider $S_1 \cup S_2$ as the sink members.

Note that a subset of non-sink members might also declare themselves as a sink. For example, in Fig. 3a, processes in $\{1, 2, 3, 4, 6\}$ that are non-sink members can declare themselves as a sink. Specifically, $\texttt{isSink}_{G_{di}}(2, \{1, 2, 3, 4, 6\}, \{5, 7\}) = true$. If that happens, the Agreement property of consensus can be violated as members of each sink can solve consensus independently of other processes.

This impossibility result indicates that a new type of knowledge connectivity graph is required for processes to solve consensus using only their PDs and without knowing f. In the next section, we achieve this by modifying a knowledge connectivity graph that meets the BFT-CUP model's requirements by adding new edges or removing existing ones.

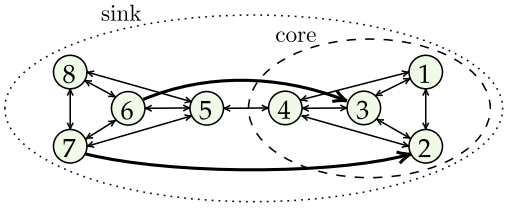
V. The BFT-CUPFT Model

The primary challenge in solving consensus in the BFT-CUPFT model, when the initial knowledge of processes collectively forms a knowledge connectivity graph belonging to $\mathcal{G}_{di}$, lies in the possibility of (a) existing multiple subsets of processes, each identifying itself as a sink, and (b) having some correct processes that cannot identify whether they are sink or non-sink members, thereby they cannot solve consensus. Therefore, to enable processes to solve consensus in the BFT-CUPFT model, it is crucial to prevent the emergence of multiple sinks, along with ensuring that each process can identify whether it is a sink or non-sink member. This objective is accomplished in this section by adding new edges to, or removing the existing ones from, the knowledge connectivity graphs belonging to $\mathcal{G}_{di}$, thereby creating a new type of knowledge connectivity graph. Before introducing such a graph, we present a few definitions.

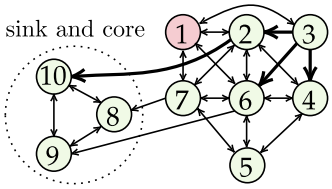
Defining a sink without a known fault threshold. Recall from Observation 1 that when processes lack information about the fault threshold, if there exists a natural number g, along with two subsets of processes S_1 and S_2, such that $\texttt{isSink}_{G_{di}}(g, S_1, S_2) = true$, then processes in S_1 consider set $S_1 \cup S_2$ as members of a sink. When the fault threshold is unknown, we define a subset of processes S as a sink if:

$$\texttt{isSink}^*_{G_{di}}(S) \iff \exists g \geq 0, \exists S_1, S_2 \subseteq S : S_1 \cup S_2 = S \wedge \texttt{isSink}_{G_{di}}(g, S_1, S_2).$$

When a set $S = S_1 \cup S_2$ is identified as a sink, we denote $\texttt{f}_{G_{di}}(S)$ as the maximum value of g that satisfies $\texttt{isSink}_{G_{di}}(g, S_1, S_2)$. Additionally, $\texttt{k}_{G_{di}}(S) = \texttt{f}_{G_{di}}(S) + 1$ represents the connectivity of S.



(a) In this knowledge connectivity graph, the sink component differs from the core component.



(b) In this knowledge connectivity graph, the sink component is the same as the core component.

Fig. 4: Two knowledge connectivity graphs that satisfy the requirements of the BFT-CUPFT model.

New type of knowledge connectivity graph. We define a new type of knowledge connectivity graph in which multiple subsets of processes might consider themselves as sinks. However, only members of a distinguished sink called the *core* can solve consensus.

Definition 2 (Extended k-OSR PD)**.** A knowledge connectivity graph $G_{di} = (V_{di}, E_{di})$ belongs to extended k-OSR PD if it satisfies the following properties:

- It belongs to k-OSR PD.
- There is a subgraph $G_{core} = (V_{core}, E_{core})$, namely the *core*, that satisfies the following properties:
  - C1) The core has the maximum connectivity among the sinks. Specifically, if any subset of processes other than the core members is considered as a sink, then its connectivity is less than the core's connectivity, i.e., $\forall V \subseteq V_{di} : V \neq V_{core} \wedge \texttt{isSink}^*_{G_{di}}(V) \Rightarrow \texttt{k}_{G_{di}}(V_{core}) > \texttt{k}_{G_{di}}(V)$.
  - C2) From any process $i \notin V_{core}$ to any process $j \in V_{core}$ there are at least $\texttt{k}_{G_{di}}(V_{core})$ node-disjoint paths.

We now provide an insight into the properties of the extended k-OSR PD:

- Property C1 ensures that there is only one core, i.e., multiple sinks cannot declare themselves as a core in a graph belonging to the extended k-OSR PD. Indeed, the core can be uniquely identified due to its maximum connectivity among the sinks. Furthermore, this property ensures that $\mathtt{k}_{G_{di}}(V_{core}) \geq k$. Since a graph belonging to the extended k-OSR PD belongs also to the k-OSR PD, the graph has at least one sink with connectivity k. As there is no sink with a connectivity greater than the core, $\mathtt{k}_{G_{di}}(V_{core}) \geq k$.
- Property C2 ensures that the core is inside the sink of G_{di}. Note that if the core is outside of the sink of G_{di}, then the correct sink members of G_{di} must have at least one path to the core due to this property, which is not possible. Additionally, this property ensures that non-core members can discover the core, like the non-sink members that discover the sink in a graph belonging to k-OSR PD.

When a knowledge connectivity graph satisfies the following two properties, we say that it satisfies the requirements of the BFT-CUPFT model: (a) its safe subgraph G_{safe} belongs to the extended $(f+1)$-OSR PD, and (b) the core component of G_{safe} contains at least $2f+1$ processes.

The graphs depicted in Fig. 4 are examples of knowledge connectivity graphs that satisfy the requirements of the BFT-CUPFT. The graph of Fig. 4a shows an example of how processes in $\{5,6,7,8\}$ were prevented from mistakenly identifying themselves as a sink by adding two extra links from process 6 to process 3 and from process 7 to process 2. In the resulting graph, processes in $\{5,6,7,8\}$ cannot identify themselves as a sink, as they discover the existence of processes 1, 2, 3, and 4 even if process 5 is slow. Similarly, the graph of Fig. 4b shows an example of how processes in $\{1,\ldots,7\}$ were prevented from mistakenly identifying themselves as a sink.

VI. Consensus in the BFT-CUPFT Model

This section presents a protocol that solves consensus in the BFT-CUPFT model. Recall that in Section III, we presented three algorithms – Discovery, Sink, and Consensus – to solve consensus in the authenticated BFT-CUP model. Since processes can execute the Discovery and Consensus algorithms without knowing the fault threshold, we can employ them in scenarios where the fault threshold is unknown. Since there is only one core in a knowledge connectivity graph that satisfies the requirements of the BFT-CUPFT model, we design an algorithm called Core that allows processes to discover the core component. We then use the Core algorithm instead of the Sink algorithm in the Consensus algorithm for solving consensus in the BFT-CUPFT model.

The Core algorithm in the BFT-CUPFT model. The objective of the Core algorithm in the BFT-CUPFT model is to allow each process to discover the core members. In further detail, each process expands the set of processes it knows by executing the Discovery algorithm. It executes the `discovery` task until it identifies the core component. Afterward, any correct process i terminates by returning a set that contains all core members. The following theorem determines how a process can identify whether a set of processes is the core.

Algorithm 4 The Core algorithm in the BFT-CUPFT model – process i.

function `core`()
1: **fork** `discovery`()
2: **wait until** $\exists g \geq 0, \exists S_1 \subseteq \mathcal{S}^i_{received}, \exists S_2 \subseteq \mathcal{S}^i_{known} \setminus S_1 :$ $\mathtt{isSink}_{G_{di}}(g, S_1, S_2) = \mathit{true} \wedge (\forall g' > g, \forall Q_1 \subset S_1,$ $\forall Q_2 \subseteq \mathcal{S}^i_{known} \setminus Q_1 : \mathtt{isSink}_{G_{di}}(g, Q_1, Q_2) = \mathit{false})$
3: **return** $S_1 \cup S_2$

Theorem 8. In a knowledge connectivity graph that satisfies the requirements of the BFT-CUPFT model, a subgraph with the node set V_{core} is the core if (a) $\mathtt{isSink}^*_{G_{di}}(V_{core}) = \mathit{true}$, and (b) there is no set $V \subset V_{core}$ such that $\mathtt{isSink}^*_{G_{di}}(V) = \mathit{true}$ and $\mathtt{k}_{G_{di}}(V) \geq \mathtt{k}_{G_{di}}(V_{core})$.

The Core algorithm is shown in Algorithm 4. This algorithm provides only one function, `core`. Upon initiation, the algorithm forks the `discovery` task, allowing concurrent execution of the Discovery algorithm. Process i waits until the properties specified in Theorem 8 are satisfied for a subset of processes. Specifically, it waits until there exists two sets S_1 and S_2 with the conditions specified in line 2. These conditions are analogous to the properties specified in Theorem 8. The Core algorithm satisfies the properties specified in the following theorem.

Theorem 9. In a knowledge connectivity graph that satisfies the requirements of the BFT-CUPFT model, Algorithm 4 executed by any correct process (a) eventually terminates and (b) returns all core members.

Finally, in order to solve consensus in the BFT-CUPFT model, we use the Core algorithm instead of the Sink algorithm in the Consensus algorithm outlined in Algorithm 3. The following theorem states that if a knowledge connectivity graph satisfies the requirements of the BFT-CUPFT model, it is sufficient to allow processes to solve consensus.

Theorem 10. In a knowledge connectivity graph that satisfies the requirements of the BFT-CUPFT model, processes can solve consensus by executing the Core algorithm instead of the Sink algorithm in Algorithm 3.

VII. Related Work

Consensus with Unknown Participants. The evolution of the Consensus with Unknown Participants (CUP) problem has involved a series of advancements to accommodate diverse system models. Initially, Cavin et al. [6] defined the problem for failure-free asynchronous systems, introducing a participant detector abstraction to provide initial information about system membership. The information collectively forms a knowledge connectivity graph, and that work establishes the necessary and sufficient properties that knowledge connectivity graphs must satisfy to solve the CUP problem. Subsequently,

CUP was addressed in [7] for crash-prone systems using the Perfect ($\mathcal{P}$) failure detector [31]. As implementing $\mathcal{P}$ requires synchrony, Greve and Tixeuil [8] relaxed the assumption to partial synchrony [19] by augmenting the minimum required knowledge, specifically by increasing connections in the knowledge connectivity graph. This augmentation is demonstrated to be the minimum to tolerate crash failures without imposing synchrony requirements. The latest milestone extended CUP to tolerate Byzantine failures, introducing the BFT-CUP protocol [9], [10].

Consensus in directed graphs. Somewhat similar to the CUP model, several studies explore consensus in directed graphs, e.g., [32]–[35]. Nevertheless, these investigations focus on determining the properties of the underlying communication graph to achieve consensus under diverse assumptions. For instance, Tseng and Vaidya [35] established the minimal conditions of the underlying communication graph, where a participant i can transmit messages to participant j if a directed edge from i to j exists in the graph; otherwise, i cannot send messages to j. Typically, these studies assume that the set of participants and the underlying communication graph are known to all participants. However, in the CUP model, the communication graph is complete, and the objective is to determine the necessary and sufficient initial knowledge about other participants required to solve consensus without knowing the system's membership.

Consensus on heterogeneous quorum systems. The exploration of protocols designed for systems where participants can have different trust assumptions, i.e., each participant can trust a subset of participants, originated in [36]. Ripple [37], [38] attempted to leverage this approach to address consensus in the permissionless setting, aiming to establish an efficient blockchain infrastructure. However, the achievement of this goal faced challenges leading to safety and liveness violations [39]. In contrast, Stellar [12], [14], based on the Federated Byzantine Quorum System (FBQS) formally studied later in [13], successfully achieved this objective. In this approach, a network of trust emerges from the partial view declared by each participant. Consensus in this network is ensured if it adheres to the *intact set* property, stipulating that all correct participants must form a quorum, and any two quorums formed by correct participants must intersect.

The connection between FBQS and dissemination Byzantine quorum systems [1] was established in [13] and [15], showing the construction of a dissemination Byzantine quorum system corresponding to an FBQS. Subsequent work by Losa et al. [16] generalized FBQS to Personal Byzantine Quorum System (PBQS), demonstrating that consensus with weaker properties than the intact set is achievable through a *consensus cluster*. Notably, forming a quorum by all correct participants is not mandatory within the consensus cluster.

Cachin and Tackmann [40] extended Byzantine Quorum Systems (BQS) [1] from the symmetric trust model to the asymmetric model, facilitating a comparison between PBQS and the classical BQS model. Recently, Cachin et al. extended the asymmetric trust model, allowing each participant to make assumptions about the failures of participants it knows and, through transitivity, about failures of participants indirectly known by it [17]. Li et al. [18] recently introduced heterogeneous quorum systems similar to PBQSs and demonstrated that the two properties – quorum intersection and availability – are necessary but insufficient to solve consensus. They introduced the notion of quorum subsumption and established that the three conditions together are sufficient. Furthermore, Vassantlal et al. [11] showed that the Stellar consensus protocol cannot solve consensus when each participant has only the minimum knowledge required to solve consensus, even though BFT-CUP can.

This line of research diverges from the CUP model in the following aspects. While these studies presume each process possesses a local fault threshold, the CUP model operates under the assumption of a global fault threshold. Furthermore, these studies identify the properties of quorums in order to solve consensus, whereas the CUP model outlines the requirements for knowledge connectivity graphs.

Sleepy model. The CUP model addresses consensus in partially synchronous systems, accommodating correct or faulty participants. However, this model assumes that correct members remain actively engaged throughout the entire execution, which may be impractical in real-world scenarios. In contrast, the sleepy model [41], [42] introduces a different perspective. In this model, participants in a synchronous system are categorized as either awake or asleep, with awake participants capable of being either faulty or correct. The system's fault tolerance dynamically adjusts as participants transition between awake and asleep states. Crucially, consensus can be achieved if the majority of awake participants are correct at any given time. Moreover, unlike CUP, all participants in this model have knowledge of the system's membership.

Consensus using broadcast medium. Khanchandani and Wattenhofer [21] established the impossibility of solving consensus in non-synchronous systems where participants and the fault threshold are unknown. In their system, *no process initially possesses knowledge about other processes*, and each process utilizes a broadcast medium for communication. Recall that our primary goal in this paper is to solve consensus in partially synchronous systems, where each process lacks information about the system's membership and the fault threshold. Accordingly, at first glance, our goal might appear contradictory to that impossibility result. However, each process has knowledge about the existence of a subset of processes in our model (BFT-CUPFT), which is sufficient to discover the core component. Recall that all processes discover the same core. The correct processes within the core can solve consensus and inform other processes about the decided value, resulting in solving consensus by all correct processes. Hence, that impossibility result does not apply to our work.

VIII. Conclusion

We addressed the critical challenge of solving Byzantine fault-tolerant consensus in partially synchronous systems

where each participant joins the network by having only partial knowledge about the existence of other participants and without explicit information about the fault threshold. We demonstrated that the key challenge arises from the possibility of having multiple disjoint subsets of processes, each solving a distinct instance of consensus, thereby violating the Agreement property of consensus. In order to mitigate this issue, we specified the sufficient knowledge connectivity requirements that must be satisfied to allow solving consensus in such settings.

Acknowledgments

This work was supported by FCT through the Ph.D. scholarship, ref. 2020.04412.BD, the SMaRtChain project, ref. 2022.08431.PTDC, and the LASIGE Research Unit, ref. UIDB/00408/2020 and ref. UIDP/00408/2020.

References

[1] D. Malkhi and M. Reiter, "Byzantine quorum systems," *Distributed Computing*, vol. 11, 1998.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *Proceedings of the International Workshop on Open Problems in Network Security*, 2015.

[4] K. Korkmaz, J. Bruneau-Queyreix, S. B. Mokhtar, and L. Réveillère, "Alder: Unlocking blockchain performance by multiplexing consensus protocols," in *Proceedings of the International Symposium on Network Computing and Applications*, 2022.

[5] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," in *Proceedings of the International Symposium on Distributed Computing*, 2017.

[6] D. Cavin, Y. Sasson, and A. Schiper, "Consensus with unknown participants or fundamental self-organization," in *Proceedings of the International Conference on Ad-Hoc Networks and Wireless*, 2004.

[7] ——, "Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes," EPFL - LSR, Tech. Rep., 2005.

[8] F. Greve and S. Tixeuil, "Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2007.

[9] E. Alchieri, A. Bessani, J. Silva Fraga, and F. Greve, "Byzantine consensus with unknown participants," in *Proceedings of the International Conference On Principles Of Distributed Systems*, 2008.

[10] E. Alchieri, A. Bessani, F. Greve, and J. Silva Fraga, "Knowledge connectivity requirements for solving Byzantine consensus with unknown participants," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 2, 2016.

[11] R. Vassantlal, H. Heydari, and A. Bessani, "On the minimal knowledge required for solving stellar consensus," in *Proceedings of the International Conference on Distributed Computing Systems*, 2023.

[12] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb, "Fast and secure global payments with Stellar," in *Proceedings of the Symposium on Operating Systems Principles*, 2019.

[13] Á. García-Pérez and A. Gotsman, "Federated Byzantine quorum systems," in *Proceedings of the International Conference on Principles of Distributed Systems*, 2018.

[14] D. Mazieres, "The Stellar consensus protocol: A federated model for internet-level consensus," https://stellar.org/papers/stellar-consensus-protocol.pdf, 2015.

[15] Álvaro García-Pérez and M. A. Schett, "Deconstructing Stellar consensus," in *Proceedings of the International Conference On Principles Of Distributed Systems*, 2020.

[16] G. Losa, E. Gafni, and D. Mazières, "Stellar consensus by instantiation," in *Proceedings of the International Symposium on Distributed Computing*, 2019.

[17] C. Cachin, G. Losa, and L. Zanolini, "Quorum systems in permissionless network," in *Proceedings of the International Conference On Principles Of Distributed Systems*, 2022.

[18] X. Li, E. Chan, and M. Lesani, "Quorum subsumption for heterogeneous quorum systems," in *Proceedings of the International Symposium on Distributed Computing*, 2023.

[19] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, 1988.

[20] I. Abraham, S. Devadas, K. Nayak, and L. Ren, "Brief announcement: Practical synchronous byzantine consensus," in *Proceedings of the International Symposium on Distributed Computing*, 2017.

[21] P. Khanchandani and R. Wattenhofer, "Byzantine agreement with unknown participants and failures," in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2021.

[22] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the Symposium on Operating Systems Design and Implementation*, 1999.

[23] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *Proceedings of the Symposium on Principles of Distributed Computing*, 2019.

[24] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, 1985.

[25] M. Bravo, G. Chockler, and A. Gotsman, "Making Byzantine consensus live," *Distributed Computing*, vol. 35, 2022.

[26] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions Programming Languages and Systems*, vol. 4, no. 3, 1982.

[27] J. R. Douceur, "The Sybil attack," in *Proceedings of the International Workshop on Peer-to-Peer Systems*, 2002.

[28] H. Heydari, R. Vassantlal, and A. Bessani, "Knowledge connectivity requirements for solving bft consensus with unknown participants and fault threshold (extended version)," https://arxiv.org/abs/2405.06055, 2024.

[29] M. Herlihy, "Wait-free synchronization," *ACM Transactions on Programming Languages and Systems*, vol. 13, no. 1, 1991.

[30] H. Heydari, G. Silvestre, and A. Bessani, "How hard is asynchronous weight reassignment?" in *Proceedings of the International Conference on Distributed Computing Systems*, 2023.

[31] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, 1996.

[32] M. Biely, P. Robinson, and U. Schmid, "Agreement in directed dynamic networks," in *Proceedings of the International Colloquium on Structural Information and Communication Complexity*, 2012.

[33] N. H. Vaidya, L. Tseng, and G. Liang, "Iterative approximate Byzantine consensus in arbitrary directed graphs," in *Proceedings of the Symposium on Principles of Distributed Computing*, 2012.

[34] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler, "Gracefully degrading consensus and k-set agreement in directed dynamic networks," *Theoretical Computer Science*, vol. 726, 2018.

[35] L. Tseng and N. H. Vaidya, "Fault-tolerant consensus in directed graphs," in *Proceedings of the Symposium on Principles of Distributed Computing*, 2015.

[36] I. Damgård, Y. Desmedt, M. Fitzi, and J. B. Nielsen, "Secure protocols with asymmetric trust," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, 2007.

[37] D. Schwartz, N. Youngs, A. Britto *et al.*, "The Ripple protocol consensus algorithm," https://ripple.com/files/ripple_consensus_whitepaper.pdf, 2014.

[38] B. Chase and E. MacBrough, "Analysis of the XRP ledger consensus protocol," https://arxiv.org/abs/1802.07242, 2018.

[39] I. Amores-Sesar, C. Cachin, and J. Mićić, "Security analysis of Ripple consensus," in *Proceedings of the International Conference on Principles of Distributed Systems*, 2020.

[40] C. Cachin and B. Tackmann, "Asymmetric distributed trust," in *Proceedings of the International Conference on Principles of Distributed Systems*, 2019.

[41] A. Momose and L. Ren, "Constant latency in sleepy consensus," in *Proceedings of the Conference on Computer and Communications Security*, 2022.

[42] R. Pass and E. Shi, "The sleepy model of consensus," in *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, 2017.