Invited Paper: Conditions for the Solvability of Fault-Tolerant Consensus in Asynchronous Unknown Networks *

Fabíola Greve
Computer Science Department
Federal University of Bahia (UFBA), Brasil
fabiola@dcc.ufba.br

Sébastien Tixeuil
LIP6-CNRS & INRIA France
Université Pierre et Marie Curie
tixeuil@lip6.fr

Abstract

The consensus problem is at the heart of solutions related to the development of modern reliable distributed systems. This paper studies necessary and sufficient conditions under which fault-tolerant consensus become solvable in dynamic systems and self-organizing networks. Those conditions are related to the synchrony requirements of the environment, to the connectivity of the knowledge graph constructed by the nodes in order to communicate with their peers, as well as to the knowledge about global parameters in the system, such as, the total number of participants and the maximum number of node crashes.

1 Introduction

Searching for the principles of how to master the uncertainties of dynamic networks, such as, wireless sensor, *ad hoc* and unstructured peer to peer networks, is a primary concern in order to design reliable and effective modern distributed systems. Dynamic networks define a new model of distributed systems, which has essential differences regarding the classical one, since only a partial knowledge of the system composition can be retained and the communication graph is not complete. Thus, it brings new challenges to the specification and resolution of fundamental problems.

Consensus is the most fundamental problem in reliable distributed computing [3]. Informally, a group of processes achieves consensus when those, who did not crash, reach a common decision on some value that has been proposed before. In traditional networks, when entities behave asynchronously, consensus cannot be solved, even if one of the participants is allowed to crash [5]. *Failure detector* and *leader* oracles are elegant abstractions which encapsulate the extra synchrony necessary to circumvent this impossibility [3]. Nonetheless, extra synchrony is not sufficient to solve consensus in dynamic networks. Beyond synchrony, some level of knowledge connectivity about participants in the system is required.

This paper studies the conditions to solve consensus in a dynamic system of unknown participants, subject to process crashes. In the classic model of distributed computing, global parameters are previous known and easily determined, such as the whole system composition (Π), the number of entities (n), and the maximum number of faults (f). In a dynamic system, Π and n are considered unknown, but since entities must cooperate somehow, some degree of knowledge regarding participants is expected. The *participant detector* abstraction was proposed to handle this knowledge [1]. It provides processes an initial knowledge connectivity graph G_{di} of the system composition. One can define necessary and sufficient connectivity conditions of G_{di} able to solve fault-tolerant consensus with unknown participants, namely FT-CUP. Roughly, to solve FT-CUP the G_{di} constructed preliminarily by the processes in an asynchronous system should have at most one sink component, and this condition is necessary [1, 2, 6].

There is, indeed, a trade-off between knowledge connectivity and synchrony for FT-CUP [6]. To solve FT-CUP with the minimal requirements regarding knowledge connectivity (the one sink component), it is necessary to enrich the system with the strongest requirements regarding synchrony (the perfect failure detector $\mathcal{P}$) [2]. Taking into account the weakest synchrony conditions to solve consensus (represented by the Ω and $\diamond\mathcal{S}$ failure detectors), FT-CUP can be solved as soon as additional assumptions regarding the knowledge connectivity (beyond the one sink requirement) are made [6]. This paper presents thus recent results concerning these necessary and sufficient requirements.

The remaining of the paper is organized as follows: Section 2 provides the model and problem statement; Section 3 describes abstractions to solve consensus; Section 4 and 5 presents necessary and sufficient conditions to solve FT-CUP. Section 6 provides some concluding remarks.

*Fabíola's research is supported by grants from CAPES and CNPQ, Brasil. Sébastien's research is supported by ANR grant SHAMAN.

2 Preliminaries

System Model. We consider a distributed system that consists of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, \ldots, p_n\}$. In a *known* network, Π is known to every participating process, while in an *unknown* network, a process p_i may only be aware of a subset Π_i of Π. There are no assumptions on the relative speed of processes or on message transfer delays, *i.e.* the system is *asynchronous*. Processes communicate by sending and receiving messages through a reliable underlying routing layer, in such a way that if $p_j \in \Pi_i$, then p_i can send a message reliably to p_j. A process p_i may only send a message to another process p_j if $p_j \in \Pi_i$. Of course, if a process p_i sends a message to a process p_j such that $p_i \notin \Pi_j$, upon receipt of the message, p_j may add p_i to Π_j and send a message back to p_i. A process may fail by *crashing*, *i.e.*, by prematurely or by deliberately halting; a crashed process does not recover. A process behaves correctly (*i.e.*, according to its specification) until it (possibly) crashes. By definition, a *correct* process is a process that does not crash. A *faulty* process is a process that is not correct. Let f denote the maximum number of processes that may crash in the system.

Classical Consensus in Known Networks. In the consensus problem, every process p_i *proposes* a value v_i and all correct processes *decide* on some unique value v, in relation to the set of proposed values. More precisely, the consensus is defined by the following properties [3, 5]: *(i)* Termination: every *correct* process eventually decides some value; *(ii)* Validity: if a process decides v, then v was proposed by some process; *(iii)* Agreement: no two *correct* processes decide differently.

Uniform Consensus refines the agreement property so that it is satisfied by every process in the system (be it correct or not). So, it is changed for: *(iii)* Uniform_Agreement: no two processes (*correct or not*) decide differently.

CUP (Consensus with Unknown Participants). The goal is to solve consensus in an unknown network, where processes may *not* crash;

FT-CUP (Fault-Tolerant CUP). The goal is to solve consensus in an unknown network, where up to f processes may crash.

3 Synchrony and Knowledge Connectivity for Consensus in Fault-Prone Systems

3.1 Failure Detector: a Synchrony Abstraction

A fundamental result [5] states that even if Π is known to all processes and the number of faulty processes is bounded by 1, consensus cannot be solved by a deterministic algorithm in an asynchronous system. To enable solutions, some level of synchrony must be assumed. A nice abstraction to model network synchrony is the *failure detector* [3]. A failure detector (denoted by FD) can be seen as an oracle that provides hints on crashed processes. Failure detectors can be classified according to the properties (completeness and accuracy) they satisfy. Two classes of failure detectors are of special interest: $\mathcal{P}$ and $\diamond\mathcal{S}$, because they represent, in respective, the strongest and the weakest conditions to solve the problem [4].

Perfect FD ($\mathcal{P}$). They never make mistakes. They satisfy the *perpetual strong accuracy*, stating that no process is suspected before it crashes, and the *strong completeness* property, stating that eventually, every process that crashes is permanently suspected by every correct process.

Eventually Strong FD ($\diamond\mathcal{S}$). They can make an arbitrary number of mistakes. Yet, there is a time after which some correct process is never suspected (*eventual weak accuracy*). Moreover, they satisfy the *strong completeness* property.

Leader Detector (Ω). Another approach for encapsulating eventual synchrony consists of extending the system with a *leader detector*, which is an oracle that eventually provides the same correct process identity to all processes.

It has been proved that $\diamond\mathcal{S}$ and Ω have the same computational power and that they are the weakest class of detectors allowing to solve the consensus and the uniform consensus problem in a system of known networks [4]. Relying on $\diamond\mathcal{S}$ and Ω failure detectors to solve agreement problems assumes that a majority of processes within the group never fails, i.e., $f < n/2$.

3.2 Participant Detectors: a Knowledge Connectivity Abstraction

Participant detectors (denoted by PD) are distributed oracles that provide information about which processes participate to the system [1]. They can be implemented, for example, with the use of the local broadcast facility of wireless networks to identify the neighborhood.

We denote by $i.\mathsf{PD}$ the participant detector of process p_i. When queried by p_i, $i.\mathsf{PD}$ returns a subset of processes in Π. The information provided by $i.\mathsf{PD}$ can evolve between queries. Let $i.\mathsf{PD}(t)$ be the query of process p_i at time t. This query must satisfy the two following properties:

- Information Inclusion. The information returned by the participant detector is non-decreasing over time. $p_i \in \Pi, t' \geq t : i.\mathsf{PD}(t) \in i.\mathsf{PD}(t')$
- Information Accuracy. The participant detector does not make mistakes. $\forall p_i \in \Pi, \forall t : i.\mathsf{PD}(t) \in \Pi$

The PD abstraction enriches the system with a knowledge connectivity graph. This graph is directed since

knowledge that is given by participation detectors is not necessarily bidirectional (*i.e.* if $p_j \in i.\mathsf{PD}$, then $p_i \in j.\mathsf{PD}$ does *not* necessarily hold).

Definition 1 (Knowledge Connectivity Graph) *Let $G_{di} = (V, E)$ be the directed graph representing the knowledge relation determined by the PD oracle. Then, $V = \Pi$ and $(p_i, p_j) \in E$ if and only if $p_j \in i.\mathsf{PD}$, i.e., p_i knows p_j.*

Definition 2 (Undirected Knowledge Connectivity Graph) *Let $G = (V, E)$ be the undirected graph representing the knowledge relation determined by PD. Then, $V = \Pi$ and $(p_i, p_j) \in E$ if and only if $p_j \in i.\mathsf{PD}$ or $p_i \in j.\mathsf{PD}$.*

Based on the induced knowledge connectivity graph (either G_{di} or G), several classes of participant detectors were proposed in [1, 6, 7]:

Connectivity PD (CO). The undirected graph G induced by the PD oracle is connected.

Strong Connectivity PD (SCO). The graph G_{di} induced by the PD oracle is strongly connected.

One Sink Reducibility PD (OSR). The graph G_{di} induced by the PD oracle satisfies the following conditions:
1. the undirected graph G obtained from G_{di} is connected;
2. the directed acyclic graph obtained by reducing G_{di} to its strongly connected components has exactly one sink.

k-Connectivity PD (k-CO). The undirected graph G induced by the PD oracle is k-connected.

k-Strong Connectivity PD (k-SCO). The graph G_{di} induced by the PD oracle is k-strongly connected.

k-One Sink Reducibility PD (k-OSR). The graph G_{di} induced by the PD satisfies the following conditions:
1. the undirected graph G obtained from G_{di} is connected;
2. the directed acyclic graph obtained by reducing G_{di} to its strongly connected components has exactly one sink: G_{sink};
3. the sink component G_{sink} is k-strongly connected;
4. for each p_i, p_j, such that $p_i \notin G_{sink}$ and $p_j \in G_{sink}$, then there are k-node-disjoint paths from p_i to p_j.

In [1], the CUP problem is investigated in *fault free* networks, and it is shown that *(i)* the CO participant detector is necessary to solve CUP, *(ii)* the SCO participant detector is sufficient to solve CUP, and *(iii)* the OSR participant detector is both necessary and sufficient to solve CUP. In the next sections, we present necessary and sufficient conditions able to solve FT-CUP in *fault-prone* networks.

4 Sufficient Conditions to Solve FT-CUP

In this section, we enumerate sufficient conditions able to solve FT-CUP. These are related not only to the level of synchrony and knowledge connectivity, but also with the knowledge about global parameters in the system, that is, n and f.

4.1 Rationale behind the algorithms to solve FT-CUP

In the beginning of the execution, to obtain an initial view of the system composition, each process p_i first queries its participant detector $i.\mathsf{PD}$. The participant detector is queried *exactly once* by each p_i to ensure that the partial snapshot about the initial knowledge connectivity is consistent for all nodes in the system. This snapshot defines the common knowledge connectivity graph $G_{di} = (V, E)$. Let $G_{sink} = (V_{sink}, E_{sink})$ be the sink component of G_{di}.

In the solutions proposed to solve FT-CUP, a sequence of protocols is executed by each p_i with the following objectives: (1) enlarge the knowledge about systems participants, beyond the information returned by $i.\mathsf{PD}$; (2) differentiate processes in G_{di} and identify those who can decide, that is, those who are in the sink G_{sink}. This is because nodes in G_{sink} cannot reach the nodes in the other components, but all the other nodes in G_{di} can reach the nodes in G_{sink}. (3) execute a consensus to decide for a value.

4.2 Solving FT-CUP when n is known and f is known

When the number n of participants is known, each process can run an algorithm to gather the "full" system composition Π. Moreover, this discovery algorithm can be purely asynchronous, and there is no need to make use of a failure detector oracle to ensure progress. Let us show a sequence of algorithms [7] sufficient to solve FT-CUP: GATHER and CONSENSUS.

• Algorithm 1: GATHER – Provides nodes a "complete" view of the system participants in G_{di}.

Description: Each process p_i first queries $i.\mathsf{PD}$ to obtain the initial view of the system which is stored in Π_i. Afterwards, p_i iteratively requests newly known processes in G_{di} to get knowledge improvement about the network, until a global knowledge about the system composition is acquired. Thus, p_i sends to processes in Π_i its actual view. When, for its turn, p_i receives the view Π_j from p_j, it (i) sends to newly known processes $(\Pi_j \setminus \Pi_i)$ its actual view and (ii) updates its own view with Π_j ($\Pi_i = \Pi_i \cup \Pi_j$). *Termination:* The prospection in the graph terminates when all processes in G_{di} are discovered, that is, $|\Pi_i| = n$. Algorithm thus returns the global system composition $\Pi_i = V$.

• Algorithm 2: CONSENSUS – Decides for a value.

Description: By construction, after the execution of GATHER, all correct nodes p_i in G_{di} share the same Π_i set, thus they all know each other and can run any classical *underlying consensus*, with the aid of a failure detector.

Lemma 1 *Consider a k-CO participant detector. Let $f < k < n$ be the number of nodes that may crash. Algorithm*

GATHER *satisfies the following properties :*

• Termination*: Every correct node p_i ends execution;*

• Safety*: Every correct node p_i returns a set Π_i of processes in G_{di}, such that $\Pi_i = V$.*

Sketch of proof: Since n is known, if p_i is a correct node, it will enrich its view Π_i with other processes view until $|\Pi_i| = n$. From the characteristics of k-CO, G_{di} is k-connected and its minimum degree in is k, thus every process $p_j \in G_{di}$ (either correct or faulty after the graph formation) will belong to at least k different view sets returned by the PD. Since, at most $f < k$ nodes can crash, then there is at least one correct process p_l which will have $p_j \in \Pi_l$. Moreover, in spite of f crashes, process p_i will reach p_l to gather its view and, in consequence, to know p_j and the lemma follows. □

Proposition 1 *The k-CO participant detector and the Ω failure detector are sufficient to solve uniform FT-CUP, when n and f are known, in spite of $f < k < n$ node crashes.*

Sketch of proof: Algorithm GATHER provides each process p_i with the same set Π (Lemma 1). Then, previous indulgent algorithms aiming for solving classical consensus, which are based on a priori knowledge about Π, can be used [3]. In particular, if $f < n/2$, and $k < n$, it is possible to solve FT-CUP as well uniform FT-CUP in a system enriched with both: a k-OSR PD and a Ω FD. □

4.3 Solving FT-CUP when n is unknown but f is known

When n is unknown and f is known, processes will be able to gather only a "partial" knowledge about the system composition Π. The discovery algorithm can be purely asynchronous, and there is no need to make use of a failure detector. But, since the view knowledge is partial, it will be necessary to differentiate processes in G_{di} in order to identify those who can decide, that is, those who are in G_{sink}. Let us show a sequence of algorithms [6, 7] sufficient to solve FT-CUP: COLLECT, SINK and CONSENSUS.

• Algorithm 1: COLLECT – Provides nodes a "partial" view of the system participants in G_{di}.

Description: Each process p_i first queries i.PD to obtain the initial view of the system which is stored in Π_i. Afterwards, p_i iteratively requests newly known processes in G_{di} to get knowledge improvement about the network, until no further knowledge can be acquired. Thus, p_i realizes an asynchronous prospection in G_{di} to identify reachable processes. At the beginning, p_i inquiries every neighbor $p_j \in \Pi_i$ for its actual view Π_j. As soon as new processes are discovered, that is $\Pi_j \backslash \Pi_i \neq \emptyset$, then p_i (i) inquiries these new discovered processes belonging to $\Pi_j \setminus \Pi_i$ for their view and (ii) augments its view, such that $\Pi_i = \Pi_i \cup \Pi_j$. *Termination:* The prospection in the graph terminates when a sufficient number of responses r is received from inquired processes, that is $r \geq |\Pi_i| - f$. Algorithm thus returns the partial knowledge $\Pi_i \subseteq V$ gathered by p_i.

• Algorithm 2: SINK – Differentiates processes in G_{di} and establishes who are in the sink component.

Description: Let Π_i be the view returned by COLLECT. Notice that every process $p_i \in G_{sink}$ has the same view $\Pi_i = V_{sink}$ of the system; whereas, in the other components, every process $p_j \notin G_{sink}$ has strictly more knowledge than p_i, and additionally, p_j knows the nodes in the sink, that is $\Pi_i \subset \Pi_j$. In that sense, the algorithm is composed of two phases. In an INITIAL PHASE, p_i asks every node $p_j \in \Pi_i$ if it belongs to their view, that is, if $p_i \in \Pi_j$. In a VERIFICATION PHASE, according to the responses from every p_j, p_i determines if it belongs to the sink. Thus, $p_i \notin G_{sink}$ if a negative response is received from p_j; otherwise, $p_i \in G_{sink}$ if all the responses received are positive. *Termination:* Algorithm returns *no*, if at least one negative response is received; otherwise, it returns *yes* if at least $r \geq |\Pi_i| - f$ responses are received.

• Algorithm 3: CONSENSUS – Decides for a value.

Description: After the execution of SINK, depending on whether p_i belongs or not to the sink, two behaviors are possible. For every $p_i \in G_{sink}$, an AGREEMENT PHASE is launched in order to reach a consensus on some value. By construction, all nodes in G_{sink} share the same Π set, thus they all known each other and can run a classical *underlying consensus* to achieve a decision. The other nodes $p_j \notin G_{sink}$ do not participate to this underlying consensus. They launch a REQUEST PHASE to ask for and collect the value decided by the sink members. This is done by sending request messages to known processes in Π_j, and waiting for one response. *Termination:* The algorithm ends up when a value is decided from the *underlying consensus* and all processes in G_{di} receives this value.

Lemma 2 *Consider a k-OSR participant detector. Algorithm* COLLECT *satisfies the following properties :*

• Termination*: Every correct node p_i terminates execution and returns a list Π_i of known nodes;*

• Safety*:* COLLECT *executed by a process p_i returns a set Π_i of processes reachable from p_i, (i) if $p_i \in G_{sink}$, then $\Pi_i = V_{sink}$. (ii) if $p_i \notin G_{sink}$, then $\Pi_i \supset V_{sink}$.*

Proposition 2 *The k-SCO participant detector and the Ω failure detector are sufficient to solve uniform FT-CUP, when f is known, in spite of $f < k < n$ node crashes.*

Sketch of proof: If PD $\in k$-SCO, there is exactly one k-strongly connected component in G_{di}. Thus, COLLECT provides each process p_i with the set Π (from Lemma 2), in spite of $f < k$ crashes. Then, similarly to Proposition 1,

previous indulgent algorithms aiming for solving classical consensus when Π is known can be used [3]. □

Lemma 3 *Consider a k-OSR participant detector. Algorithm* SINK *satisfies the following properties:*

- Termination*: Every correct node p_i ends execution and decides whether it belongs to G_{sink};*
- Safety*: A node p_i is in the unique sink component G_{sink}* iff *algorithm* SINK *returns* true.

Proposition 3 *The k-OSR participant detector and the Ω failure detector are sufficient to solve uniform FT-CUP, when f is known, in spite of $f < k < n$ node crashes, assuming at least $2f + 1$ correct nodes in G_{sink}.*

Sketch of proof: Te execution of COLLECT and SINK establishes which processes are in G_{sink} (Lemma 3). Then, on the execution of CONSENSUS, every process $p_i \in G_{sink}$ can run an indulgent *underlying_consensus*, as soon as at least $2f + 1$ processes are correct and the system is extended with an Ω failure detector [3]. Thus, for every $p_i \in G_{sink}$, properties Validity, Termination and Uniform Agreement come trivially from the underlying consensus properties [3]. The other processes $p_j \notin G_{sink}$ will ask for and eventually receive the decision, as soon it is taken by the nodes in G_{sink}, since moreover, there is at least one correct process in G_{sink}. □

4.4 Solving FT-CUP when n is unknown and f is unknown

When n is unknown and moreover f is unknown, to discover processes in G_{di}, in order to ensure progress, it will be necessary to enrich the system with a failure detector. Additionally, the information given by this FD should be reliable, otherwise, a process unduly suspected of being faulty is not going to be heard by the others and its view is not going to be considered. In consequence, the partial view gathered by processes will be inconsistent and the perception of the knowledge connectivity graph may diverge, compromising the safety properties of FT-CUP.

A sequence of algorithms able to solve FT-CUP, when f and n are unknown, is presented in [2]. It assumes an OSR participant detector and a perfect $\mathcal{P}$ failure detector. Moreover, it assumes a *safe crash pattern*, which imposes a restriction on the set of processes defined by both the participant and failure detectors.

Definition 3 (Safe Crash Pattern) *Let G_{di} be the common knowledge connectivity graph formed by processes in the system on their first invocation to PD. Let $\mathcal{P}.suspect$ be the list of suspected processes returned by the perfect failure detector $\mathcal{P}$. The* safe crashed pattern *is the set of processes obtained from the combination of the information returned by the PD and the FD, and such that $G_{di} \setminus \mathcal{P}.suspect \in OSR$.*

Proposition 4 *The OSR participant detector and the perfect $\mathcal{P}$ failure detector are sufficient to solve FT-CUP, when n and f are unknown, in spite of $f < n$ node crashes, assuming the safe crashed pattern ($G_{di} \setminus \mathcal{P}.suspect \in OSR$) holds at any time.*

Sketch of proof: The execution of a DISCOVERY algorithm will enlarge the view of the processes and ensure that, at least all processes S in the sink component G_{sink} which have not crashed during the execution, that is, $S = G_{sink} \setminus \mathcal{P}.suspect$, are going to be known by every other process in G_{di}. Then, the execution of a LEADER ELECTION algorithm will chose a leader, among processes in S, and this leader will finally impose its own value as the decided value. If during the execution of the algorithms a process crashes, the properties of $\mathcal{P}$ ensure that it will be eventually suspected and then progress is ensured in a safe manner. □

5 Necessary Conditions to Solve FT-CUP

5.1 When the system is not equipped with a failure detector

In this section, we depict the necessary conditions for solving FT-CUP in an asynchronous system, subject to arbitrary failures and extended only with a participant detector [6, 7, 1].

Proposition 5 *The $(f+1)$-CO participant detector is necessary to solve FT-CUP in an asynchronous unknown network, in spite of $f < n$ node crashes.*

Sketch of proof: Assume by contradiction that the undirected knowledge connectivity graph G defined by the PD oracle is f-connected. But, the removal of f nodes may disconnect this undirected graph G into at least two components C_1 and C_2. Since nodes in C_1 do not communicate with C_2, they can execute a consensus independently in each component and may decide different values, violating *agreement*. □

Proposition 6 *The OSR participant detector is necessary to solve FT-CUP in an asynchronous unknown network, in spite of $f < n$ node crashes.*

Sketch of proof: Assume by contradiction that there is an algorithm $\mathcal{A}$ which solves FT-CUP with a PD $\notin OSR$. Suppose that the decomposition of G_{di} to its strongly connected components has more than one sink. Let G_1 and G_2 be two of those sinks. Assume that all nodes in G_1 have

input value equal to v and that all nodes in G_2 have input value equal to w, $v \neq w$. By the *termination* property of consensus, nodes in G_1 decide at time $t1$ and nodes in G_2 decide at time $t2$. We can delay the reception of any messages from nodes in other components to both G_1 and G_2 to a time $t > max\{t1, t2\}$. Since nodes in the sinks are unaware about the existence of other nodes, by the *validity* of consensus, nodes in G_1 decide for the value v and nodes in G_2 decide for the value w, violating the *agreement*. □

Proposition 7 *The $(f+1)$-OSR participant detector is necessary to solve FT-CUP in an asynchronous unknown network, in spite of $f < n$ node crashes.*

Sketch of proof: Assume by contradiction that there is an algorithm $\mathcal{A}$ which solves FT-CUP with a PD $\notin (f+1)$-OSR. Suppose that (1) G obtained from G_{di} is not connected, or (2) the decomposition of G_{di} has more than one sink, or (3) the sink G_{sink} is not $(f+1)$-strongly connected or (4) there are p_i, p_j, such that $p_i \notin G_{sink}$ and $p_j \in G_{sink}$, then there exists less than $(f+1)$-node-disjoint paths from p_i to p_j. In scenarios (1), (2) and (3) the occurrence of f crashes may either disconnect the sink into at least two disjoint components or generate more than one sink. But, connectivity of G and OSR are necessary conditions (Propositions 5 and 6). In scenario (4), from Proposition 6, decision must be taken in the sink; otherwise, *agreement* is infringed. Thus, if $p_i \notin G_{sink}$, p_i must wait from a decision coming from a reachable $p_j \in G_{sink}$, $p_i \rightsquigarrow p_j$. But, the removal of f nodes, may disconnect p_i from p_j, and the decision will never arrive at p_i, compromising *termination*. □

5.2 When the system is equipped with a failure detector

When the system is enriched with a failure detector oracle, it is possible to weaken the knowledge connectivity conditions and the knowledge about f. But, the information about failures should be reliable. As stated before, admitting false suspicions may result in creating different perceptions of the knowledge connectivity graph and the violation of FT-CUP properties [2].

Proposition 8 *The following conditions: (1) OSR participant detector, (2) $\mathcal{P}$ failure detector and (3) safe crash pattern ($G_{di} \setminus \mathcal{P}.suspect \in OSR$) are necessary to solve FT-CUP in an asynchronous unknown network, in spite of $f < n$ node crashes.*

Sketch of proof: Let $i.PD$ the view returned by the PD of p_i and $i.suspect$ the list of suspects returned by the FD associated to p_i. Condition (1) follows from Proposition 6. To prove Condition (2) assume, by contradiction, that the FD does not satisfy the *strong completeness* property. Then, there may be a crashed process p_j, known by a correct process p_i, $p_j \in i.PD$, but which is never suspected by p_i, $p_j \notin i.suspected$. Suppose that $G_{sink} = (\{p_j\}, \emptyset)$. In this case, to ensure agreement, p_i should wait for p_j's response. But, this response will never come, since p_j is faulty. Thus, *termination* is not ensured, reaching a contradiction. Now, assume that the FD does not satisfy the *strong accuracy* property. Then, correct processes are unduly suspected. Let p_i and p_j be two correct processes, such that, $p_i \in j.PD$ and $p_i \in j.suspect$, $p_j \in i.PD$ and $p_j \in i.suspect$. In this case, the false suspicion can lead both processes to be partitioned into two non-intersection sinks; one formed by $\{p_i\}$ and another by $\{p_j\}$. From Proposition 6, decision must be taken in a unique sink component; otherwise, *agreement* is infringed, reaching a contradiction. Condition (3) follows from Condition (1) and Condition (2). □

6 Conclusion

In this paper, we provide necessary and sufficient conditions able to solve consensus in fault-prone unknown networks. These conditions are related to (i) the information returned by the participant detector concerning the knowledge connectivity formed by processes in the system (ii) the synchrony assumptions encapsulated by a failure detector; (iii) the knowledge about global parameters: n and f.

References

[1] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In *Proc. 3rd Int. Conf. AD-NOC Networks & Wireless (ADHOC-NOW)*, pages 135–148, Vancouver, July 2004. Springer-Verlag.

[2] D. Cavin, Y. Sasson, and A. Schiper. Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Research Report IC/2005/026, EPFL, 2005.

[3] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, Mar. 1996.

[4] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[5] M. J. Fischer, N. A. Lynch, and M. D. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382, Apr. 1985.

[6] F. Greve and S. Tixeuil. Knowledge Connectivity vs. Synchrony Requirements for Fault-Tolerant Agreement in Unknown Networks. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 82–91, Washington, DC, USA, 2007. IEEE Computer Society.

[7] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. Technical report, INRIA, Paris, France, 2010.