A Necessary and Sufficient Synchrony Condition for Solving Byzantine Consensus

Olivier Baldellon, Achour Mostefaoui, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1954 – juillet 2010

A Necessary and Sufficient Synchrony Condition for Solving Byzantine Consensus

Olivier Baldellon* , Achour Mostefaoui** , Michel Raynal***
olivier.baldellon@irisa.fr, achour@irisa.fr, raynal@irisa.fr

Abstract: Solving the consensus problem requires in one way or another that the underlying system satisfies synchrony assumptions. Considering a system of n processes where up to $t < n/3$ may commit Byzantine failures, this paper investigates the synchrony assumptions that are required to solve consensus. It presents a corresponding necessary and sufficient condition.

Such a condition is formulated with the notions of a synchrony property and property ambiguity. A synchrony property is a set of graphs, where each graph corresponds to a set of eventually synchronous links among correct processes. Intuitively, a property is ambiguous if it contains a graph whose connected components are such that it is impossible to distinguish a connected component that contains correct processes only from a connected component that contains faulty processes only. The paper connects then the notion of a synchrony property with the notion of eventual bi-source, and shows that the existence of a virtual $\diamond[t+1]$bi-source is a necessary and sufficient condition to solve consensus in presence of up to t Byzantine processes in systems with message authentication.

Key-words: Asynchronous message-passing system, Byzantine process, Consensus problem, Eventually synchronous link, Fault-tolerance, Lower bound, Signature, Synchrony property.

Une condition nécessaire et suffisante pour le consensus byzantin

Résumé : *Ce rapport prouve une condition nécessaire et suffisante sur la synchronie des canaux de communication qui permet de résoudre le consensus byzantin avec authentification. En effet, si les canaux sont inluctablement synchrones le consensus peut être résolu et si les canaux sont totalement asynchrones le problème de consensus est indécidable. Cet article montre qu'il est nécessaire et suffisant d'avoir $4t$ canaux inéluctablement synchrones pour résoudre le consensus byzantin avec authentification, t étant le nombre maximal de processus byzantins.*

Mots clés : *Système à communication par messages, processus byzantin, problème de consensus, tolèrance aux fautes, canaux synchrones authentification.*

---

* Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes
** Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes
*** Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes

1 Introduction

Byzantine consensus A process has a *Byzantine* behavior when it behaves arbitrarily [18]. This bad behavior can be intentional (malicious behavior, e.g., due to intrusion) or simply the result of a transient fault that altered the local state of a process, thereby modifying its behavior in an unpredictable way.

We are interested here in the *consensus* problem in distributed systems prone to Byzantine process failures whatever their origin. Consensus is an agreement problem in which each process first proposes a value and then decides on a value [18]. In a Byzantine failure context, the consensus problem is defined by the following properties: every non-faulty process decides a value (termination), no two non-faulty processes decide different values (agreement), and if all non-faulty processes propose the same value, that value is decided (validity). (See [17] for a short introduction to Byzantine consensus.)

Aim of the paper A synchronous distributed system is characterized by the fact that both processes and communication links are synchronous (or timely) [2, 15, 20]. This means that there are known bounds on process speed and message transfer delays. Let t denote the maximum number of processes that can be faulty in a system made up of n processes. In a synchronous system, consensus can be solved (a) for any value of t (i.e., $t < n$) in the crash failure model, (b) for $t < n/2$ in the general omission failure model, and (c) for $t < n/3$ in the Byzantine failure model [14, 18]. Moreover, these bounds are tight.

On the contrary, when all links are asynchronous (i.e., when there is no bound on message transfer delays), it is impossible to solve consensus even if we consider the weakest failure model (namely, the process crash failure model) and assume that at most one process may be faulty (i.e., $t = 1$) [9]. It trivially follows that Byzantine consensus is impossible to solve in an asynchronous distributed system.

As Byzantine consensus can be solved in a synchronous system and cannot in an asynchronous system, a natural question that comes to mind is the following "When considering the synchrony-to-asynchrony axis, which is the weakest synchrony assumption that allows Byzantine consensus to be solved?" This is the question addressed in the paper. To that end, the paper considers the synchrony captured by the structure and the number of eventually synchronous links among correct processes.

Related work Several approaches to solve Byzantine consensus have been proposed. We consider here only deterministic approaches[1]. One consists in enriching the asynchronous system (hence the system is no longer fully asynchronous) with a failure detector, namely, a device that provides processes with hints on failures [5]. Basically, in one way or another, a failure detector encapsulates synchrony assumptions. Failure detectors suited to Byzantine behavior have been proposed and used to solved Byzantine consensus (e.g., [4, 7, 10, 11, 16]).

Another approach proposed to solve Byzantine consensus consists in directly assuming that some links satisfy a synchrony property ("directly" means that the synchrony property is not hidden inside a failure detector abstraction). This approach relies on the notion of a $\diamond[x + 1]$*bi-source* (read "$\diamond$" as "eventual") that has been introduced in [1]. Intuitively, this notion states that there is a correct process that has x bi-directional input/outputs links with other correct processes and these links eventually behave synchronously [6, 8]. (Our definition of a $\diamond[x + 1]$bi-source is slightly different from the original definition introduced in [1]. The main difference is that it considers only eventual synchronous links connecting correct processes. It is precisely defined in Section 6[2].)

Considering asynchronous systems with Byzantine processes without message authentication, it is shown in [1] that Byzantine consensus can be solved if the system has a $\diamond[n - t]$bi-source (all other links being possibly fully asynchronous). Moreover, the $\diamond[n - t]$bi-source can never be explicitly known. This result has been refined in [13] where is presented a Byzantine consensus algorithm for an asynchronous system that has a $\diamond[2t + 1]$bi-source. Considering systems with message authentication, a Byzantine consensus algorithm is presented in [12] that requires a $\diamond[t + 1]$bi-source only. As for Byzantine consensus in synchronous systems, all these algorithms assume $t < n/3$.

Content of the paper The contribution of the paper is the definition of a synchrony property that is necessary and sufficient to solve Byzantine consensus in asynchronous systems with message authentication. From a concrete point of view, this property is the existence of what we call a virtual $\diamond[t + 1]$bi-source.

A *synchrony* property S is a set of communication graphs, such that (a) each graph specifies a set of eventually synchronous bi-directional links connecting correct processes and (b) this set of graphs satisfies specific additional properties that give S a particular structure. A synchrony property can be or not *ambiguous*. Intuitively, it is ambiguous if it contains a graph whose connected components are such that there are executions in which it is impossible to distinguish a component with correct processes only from a connected component with faulty processes only. (These notions are formally defined in the paper).

A *synchrony* property S for a system of n processes where at most t processes may be faulty is called (n, t)-synchrony property. The paper shows first that, assuming a property S, it is impossible to solve consensus if S is ambiguous. It is then shown that, if consensus can be solved when the actual communication graph is any graph of S (we then say "S is satisfied"), then any graph of S has at least

[1] Enriching the system with random numbers allows for the design of randomized Byzantine consensus algorithms. These algorithms are characterized by a probabilistic termination property (e.g., [3, 19, 21].

[2] We consider eventually synchronous links connecting correct processes only for the following reason. This is because, due to Byzantine behavior, a synchronous link connecting a correct process and a Byzantine process can always appear to the correct process as being an asynchronous link.

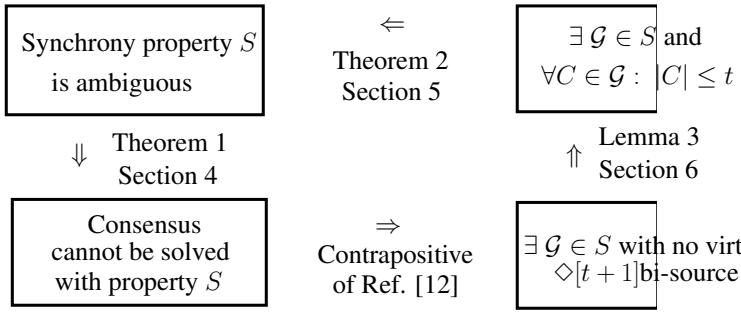

Figure 1: The proof of the necessary and sufficient condition (Theorem 3)

one connected component whose size is at least $t + 1$. The paper then relates the ambiguity of an (n, t)-synchrony property S with the size x of a virtual $\diamond[x]$bi-source. These results are schematically represented in Figure 1 from which follows the fact that a synchrony property S allows Byzantine consensus to be solved despite up to t Byzantine processes in a system with message authentication if and only if S is not ambiguous.

Road map The paper is made up of 6 sections. Section 2 presents the underlying asynchronous Byzantine computation model. Section 3 defines the notion of a synchrony property S and the associated notion of ambiguity. As already indicated, a synchrony property is on the structure of eventually synchronous links connecting correct processes. Then, Section 4 shows that an ambiguous synchrony property S does not allow consensus to be solved (Theorem 1). Section 5 relates the size of connected components of the graphs of an (n, t)-synchrony property S with the ambiguity of S (Theorem 2). Section 6 establishes the main result of the paper, namely a necessary and sufficient condition for solving Byzantine consensus in system with message authentication.

2 Computation model

Processes The system is made up of a finite set $\Pi = \{p_1, \ldots, p_n\}$ of $n > 1$ processes that communicate by exchanging messages through a communication network. Processes are assumed to be synchronous in the sense that local computation times are negligible with respect to message transfer delays. Local processing times are considered as being equal to 0.

Failure model Up to $t < n/3$ processes can exhibit a *Byzantine* behavior. A Byzantine process is a process that behaves arbitrarily: it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transition, etc. Moreover, Byzantine processes can collude to "pollute" the computation. Yet, it is assumed that they do not control the network. This means that they cannot corrupt the messages sent by non-Byzantine processes, and the schedule of message delivery is uncorrelated to Byzantine behavior. A process that exhibits a Byzantine behavior is called *faulty*. Otherwise, it is *correct* or *non-faulty*.

Communication network Each pair of processes p_i and p_j is connected by a reliable bi-directional link denoted (p_i, p_j). This means that, when a process receives a message, it knows which is its sender.

A link can be fully asynchronous or eventually synchronous. The bi-directional link connecting a pair of processes p_i and p_j is *eventually synchronous* if there is a finite (but unknown) time τ after which there is an upper bound on the time duration that elapses between the sending and the reception of a message sent on that link (hence an eventually synchronous link is eventually synchronous in both directions). If such a bound does not exist the link is fully asynchronous. If $\tau = 0$ and the bound is known, then the link is synchronous.

Message authentication When the system provides the processes with message authentication, a Byzantine process can fail to relay messages or send bad messages only. When it forwards a message received from another process it cannot alter its content.

Notation Let $\mathcal{H} \subseteq \Pi \times \Pi$ denote the communication graph whose edges are the eventually synchronous bi-directional links among correct processes. The previous system model is denoted $\mathcal{AS}_{n,t}[\mathcal{H}]$.

3 Definitions

We consider only undirected graphs in the following. The aim of this section is to state a property that will be used to prove an impossibility result. Intuitively, a vertex represents a process, while an edge is used to represent an eventually synchronous bi-directional link. Hence the set of vertices of a graph $\mathcal{G}$ is Π and its set of edges is included in $\Pi \times \Pi$.

3.1 (n, x)-Synchrony property and ambiguity

The formal definitions given in this section will be related to processes and links of a system in the next section.

Definition 1. *Let $\mathcal{G} = (\Pi, E)$ be a graph. A permutation π on Π defines a* permuted graph*, denoted $\pi(\mathcal{G}) = (\Pi, E')$, where $\forall\ a, b \in \Pi :\ \big((a, b) \in E\big) \Leftrightarrow \big((\pi(a), \pi(b)) \in E'\big)$.*

All permuted graphs of $\mathcal{G}$ have the same structure as $\mathcal{G}$, they differ only in the names of vertices.

Definition 2. *Let $\mathcal{G}_1 = (\Pi, E1)$ and $\mathcal{G}_2 = (\Pi, E2)$. $\mathcal{G}_1$ is* included *in $\mathcal{G}_2$ (denoted $\mathcal{G}_1 \subseteq \mathcal{G}_2$) if $E1 \subseteq E2$.*

Definition 3. *An (n, x)-*synchrony *property S is a set of graphs with n vertices such that $\forall \mathcal{G}_1 \in S$ we have:*

- Permutation stability. *If $\mathcal{G}_2$ is a permuted graph of $\mathcal{G}_1$, then $\mathcal{G}_2 \in S$.*
- Inclusion stability. *$\forall\ \mathcal{G}_2$ such that $\mathcal{G}_1 \subseteq \mathcal{G}_2$ then $\mathcal{G}_2 \in S$.*
- x-Resilience. *$\exists\ \mathcal{G}_0 \in S$ such that $\mathcal{G}_0 \subseteq \mathcal{G}_1$ and $\mathcal{G}_0$ has at least x isolated vertices*[3].

The aim of an (n, x)-synchrony property is to capture a property on eventually synchronous bi-directional links. It is independent from process identities (permutation stability). Moreover, adding eventually synchronous links to a graph of an (n, x)-synchrony property S does not falsify it (inclusion stability). Finally, the fact that up to x processes are faulty cannot invalidate it (x-resilience).

As an example, assuming $n - t \geq 3$, "there are 3 eventually synchronous bi-directional links connecting correct processes" is an (n, x)-synchrony property. It includes all the graphs $\mathcal{G}$ of n vertices that have 3 edges and x isolated vertices plus, for every such $\mathcal{G}$, all graphs obtained by adding any number of edges to $\mathcal{G}$.

Given a graph $\mathcal{G} = (\Pi, E)$ and a set of vertices $C \subset \Pi$, $\mathcal{G} \setminus C$ denotes the graph from which edges (p_i, p_j) with p_i or $p_j \in C$ have been removed.

Definition 4. *Let S be an (n, x)-synchrony property. S is* ambiguous *if it contains a graph $\mathcal{G} = (\Pi, E)$ whose every connected component C is such that (i) $|C| \leq x$ and (ii) $\mathcal{G} \setminus C \in S$. Such a graph $\mathcal{G}$ is said to be S-*ambiguous*.*

Intuitively, an (n, x)-synchrony property S is ambiguous if it contains a graph $\mathcal{G}$ that satisfies the property S in all runs where all processes of any connected component of $\mathcal{G}$ are faulty (recall that at most x processes are faulty).

3.2 Algorithm and runs satisfying an (n, x)-synchrony property

Definition 5. *An n-process* algorithm $\mathcal{A}$ *is a set of n deterministic automata, one for each process. An execution of $\mathcal{A}$ is a sequence of steps issued by processes. A step corresponds to an atomic action. During a step a process may send/receive a message and change its state.*

Definition 6. *A* run of an algorithm $\mathcal{A}$ *is a tuple $\langle t, I, R, T, \mathcal{G}\rangle$ where t is an upper bound on the number of faulty processes, I defines the initial state of each process, R is a (possibly infinite) sequence of steps, T an increasing sequence of time values indicating the time instants at which the steps of R occurred and for any edge (p_i, p_j) of $\mathcal{G}$, both p_i and p_j are correct and the bi-directional link connecting them is eventually synchronous.*

The sequence R is such that, for any message m, the reception of m occurs after its sending and the steps issued by every process occur in R in their issuing order, and for any correct process p_i the steps of p_i are as defined by its automaton.

Definition 7. *Given a graph $\mathcal{G} = (\Pi, E)$ defining which are the eventually synchronous links among correct processes, and an algorithm $\mathcal{A}$, $\mathcal{E}(t, \mathcal{A}, \mathcal{G})$ denotes the set of runs of $\mathcal{A}$ in which at most t processes are faulty.*

Definition 8. *Given an (n, x)-synchrony property S, let $\mathcal{E}_S(\mathcal{A})$ be the set of runs $r = \langle t, I, R, T, \mathcal{G}\rangle$ of $\mathcal{A}$ such that $\mathcal{G} \in S$ and $t \leq x$. Let us observe that $\mathcal{E}_S(\mathcal{A}) = \bigcup_{\mathcal{G}\in S} \mathcal{E}(t, \mathcal{A}, \mathcal{G})$ with $t \leq x$.*

Definition 9. *An (n, x)-synchrony property S* allows an algorithm $\mathcal{A}$ to solve the consensus problem in presence of up to x faulty processes *if every run in $\mathcal{E}_S(\mathcal{A})$ satisfies the validity, agreement and termination properties that define the Byzantine consensus problem.*

[3] An isolated vertex is a vertex with no neighbor.

4 An impossibility result

Given an (n,t)-synchrony property S, this section shows that there is no algorithm $\mathcal{A}$ that solves the consensus problem in $\mathcal{AS}_{n,t}[\mathcal{H}]$ if $\mathcal{H}$ is an S-ambiguous graph of S. This means that the synchrony assumptions captured by S are not powerful enough to allow consensus to be solved despite up to t faulty processes. There is no algorithm $\mathcal{A}$ that would solve consensus for any underlying synchrony graph of an ambiguous synchrony property S.

4.1 A set of specific runs

This section defines the set of runs in which the connected components (as defined by the eventually synchronous communication graph $\mathcal{H}$) are asynchronous the ones with respect to the others, and (if any) the set of faulty processes corresponds to a single connected component. The corresponding set of runs, denoted $\mathcal{F}(\mathcal{A},\mathcal{H})$, will then be used to prove the impossibility result.

Definition 10. *Let $\mathcal{A}$ be an n-process algorithm $\mathcal{A}$ and $\mathcal{H}$ be a graph whose n vertices are processes and every connected component contains at most t processes. Let $\mathcal{F}(\mathcal{A},\mathcal{H})$ be the set of runs of $\mathcal{A}$ that satisfy the following properties:*

- *If p_i and p_j belong to the same connected component of $\mathcal{H}$, then the bi-directional link (p_i,p_j) is eventually synchronous.*
- *If p_i and p_j belong to the same connected component of $\mathcal{H}$, then both are either correct or faulty.*
- *If p_i and p_j belong to distinct connected components of $\mathcal{H}$, then the bidirectional link (p_i,p_j) is asynchronous.*
- *If p_i and p_j belong to distinct connected components of $\mathcal{H}$, then, if p_i is faulty, p_j is correct.*

4.2 An impossibility

Let S be an ambiguous (n,t)-synchrony property, $\mathcal{A}$ be an algorithm and $\mathcal{H}$ be the graph defining the eventually synchronous links among processes. The lemma that follows states that, if $\mathcal{H}$ is S-ambiguous, all runs r in $\mathcal{F}(\mathcal{A},\mathcal{H})$ belong to $\mathcal{E}_S(\mathcal{A})$.

Lemma 1. *Let S be an (n,t)-synchrony property and $\mathcal{H}\in S$. If $\mathcal{H}$ is S-ambiguous, then $\mathcal{F}(\mathcal{A},\mathcal{H})\subseteq\mathcal{E}_S(\mathcal{A})$.*

Proof As it is S-ambiguous, $\mathcal{H}$ contains only connected components with at most t processes. It follows that the set $\mathcal{F}(\mathcal{A},\mathcal{H})$ is well-defined. Let $r\in\mathcal{F}(\mathcal{A},\mathcal{H})$.

Let $C_1,\ldots,C_m$ be the connected components of $\mathcal{H}$. We can then define $\mathcal{H}_0=\mathcal{H}$ (when no process are faulty) and for any i with $1\leq i\leq m$, $\mathcal{H}_i=\mathcal{H}\setminus C_i$ (when the set of faulty processes correspond to C_i). If in run r, all processes are correct, we have $r\in\mathcal{E}(t,\mathcal{A},\mathcal{H})$. Moreover, if there is a faulty process in run r, by defintion of $\mathcal{F}(\mathcal{A},\mathcal{H})$, the set of faulty processes correspond to a connected component. Let C_i be this connected component. We then have $r\in\mathcal{E}(t,\mathcal{A},\mathcal{H}_i)$.

We just showed that $\mathcal{F}(\mathcal{A},\mathcal{H})\subseteq\bigcup_{0\leq i\leq m}\mathcal{E}(t,\mathcal{A},\mathcal{H}_i)$. As $\mathcal{H}$ is S-ambiguous, for any $1\leq i\leq m$ we have $\mathcal{H}_i\in S$, and thanks to the inclusion stability property of S, we also have $\mathcal{H}_0=\mathcal{H}\in S$. Finally, as $\mathcal{E}_S(\mathcal{A})=\bigcup_{\mathcal{X}\in S}\mathcal{E}(t,\mathcal{A},\mathcal{X})$ we have $\mathcal{F}(\mathcal{A},\mathcal{H})\subseteq\mathcal{E}_S(\mathcal{A})$ which prove the lemma. $\square_{Lemma\ 1}$

Lemma 2. *Let S be an ambiguous (n,t)-synchrony property and $\mathcal{H}$ an S-ambiguous graph. Whatever the algorithm $\mathcal{A}$, there is a run $r\in\mathcal{F}(\mathcal{A},\mathcal{H})$ that does not solve consensus.*

Proof The proof is a reduction to the FLP impossibility result [9] (impossibility to solve consensus despite even only one faulty process in a system in which all links are asynchronous). To that end, let us assume by contradiction that there is an algorithm $\mathcal{A}$ that solves consensus among n processes $p_1,\ldots,p_n$ despite the fact that up to t of them may be faulty, when the underlying eventually synchronous communication graph belongs to S (for example an S-ambiguous graph $\mathcal{H}$). This means that, by assumption, all runs $r\in\mathcal{E}_S(\mathcal{A})$ satisfy the validity, agreement and termination properties that define the Byzantine consensus problem.

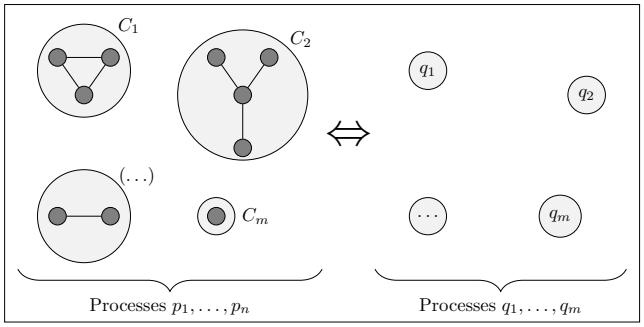

Figure 2: A reduction to the FLP impossibility result

Let $C_1, \ldots, C_m$ be the connected components of $\mathcal{H}$ and $q_1, \ldots, q_m$ a set of m processes (called simulators in the following). The proof consists in constructing a simulation in which the simulators $q_1, \ldots, q_m$ solve consensus despite the fact they are connected by asynchronous links and one of them may be faulty, thereby contradicting the FLP result (Figure 2). To that end, each simulator q_j, $1 \leq j \leq m$, simulates the processes of the connected component C_j it is associated with. Moreover, without loss of generality, let us assume that, for every component C_j made up of correct processes, these processes propose the same value v_j.

Such a simulation[4] of the processes $p_1, \ldots, p_n$ (executing the Byzantine consensus algorithm $\mathcal{A}$) by the simulators $q_1, \ldots, q_m$ results in a run $r \in \mathcal{F}(\mathcal{A}, \mathcal{H})$ (from the point of view of the processes $p_1, \ldots, p_n$). As (by definition) the algorithm $\mathcal{A}$ is correct, the correct processes decide in run r. As (a) $\mathcal{H} \in S$, (b) S is ambiguous, and (c) $r \in \mathcal{F}(\mathcal{A}, \mathcal{H})$, it follows from Lemma 1 that $r \in \mathcal{E}_S(\mathcal{A})$, which means that r is a run in which the correct processes decide the same value v (and, if they all have proposed the very same value w, we have $v = w$).

It follows that, simulating the processes $p_1, \ldots, p_n$ that execute the consensus algorithm $\mathcal{A}$, the m asynchronous processes $q_1, \ldots, q_m$ (q_j proposing value v_j) solves consensus despite the fact that one of them is faulty, contradicting the FLP impossibility result, which concludes the proof of the theorem. $\square_{Lemma\ 2}$

The following theorem is an immediate consequence of lemmas 1 and 2.

Theorem 1. *No ambiguous (n, t)-synchrony property allows Byzantine consensus to be solved in a system of n processes where up to t processes can be faulty.*

Remark 1 Let us observe that the proof of the previous theorem does not depend on the fact that messages are signed or not. Hence, the theorem is valid for both systems with and systems without message authentication.

Remark 2 The impossibility to solve consensus despite even only one process failure in an asynchronous system [9] corresponds to the case where S is the $(n, 1)$-synchrony property that contains the edge-less graph.

5 Relating the size of connected components and ambiguity

Assuming a system with message authentication, let S be an (n, t)-synchrony property that allows consensus to be solved despite up to t Byzantine processes. This means that consensus can be solved for any eventually synchronous communication graph in S. It follows from Theorem 1 that S is not ambiguous. This section shows that if an eventual synchrony property S allows consensus to be solved, then any graph of S contains at least one connected component C whose size is greater than t ($|C| > t$).

Theorem 2. *Let S be an (n, t)-synchrony property. If there is a graph $\mathcal{G} \in S$ such that none of its connected components has more than t vertices, then S is ambiguous.*

Proof Let $\mathcal{G} \in S$ such that no connected component of $\mathcal{G}$ has more than t vertices. It follows from the t-resilience property of S that there is a graph $\mathcal{G}'$ included in $\mathcal{G}$ (i.e., both have the same vertices and the edges of $\mathcal{G}'$ are also in $\mathcal{G}$) that has at least t isolated vertices.

Let us observe that $\mathcal{G}'$ can be decomposed into $m + t$ connected components $C_1, \ldots, C_m, \gamma_1, \ldots, \gamma_t$ where each C_i contains at most t vertices and each γ_i contains a single vertex (top of Figure 3).

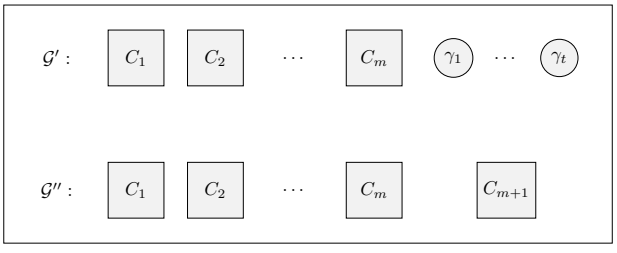

Figure 3: Construction of the graph $\mathcal{G}''$

Let us construct a graph $\mathcal{G}''$ as follows. $\mathcal{G}''$ is made up of the m connected components $C_1, \ldots, C_m$ plus another connected component denoted C_{m+1} including the t vertices $\gamma_1, \ldots, \gamma_t$ (bottom of Figure 3). Moreover, $\mathcal{G}''$ contains all edges of $\mathcal{G}'$ plus the new edges needed in order that the connected component C_{m+1} be a clique (i.e., a graph whose any pair of distinct vertices is connected by an edge). As $\mathcal{G}' \in S$ and $\mathcal{G}' \subseteq \mathcal{G}''$, it follows from the stability property of S that $\mathcal{G}'' \in S$.

The rest of the proof consists in showing that $\mathcal{G}''$ is S-ambiguous (from which ambiguity of S follows).

[4]The simulation, which is only sketched, is a very classical one. A similar simulation is presented in [18], in the context of synchronous systems, that extends the impossibility to solve Byzantine consensus from a set of $n = 3$ synchronous processes where one ($t = 1$) is a Byzantine process to a set of $n \leq 3t$ processes. A similar simulation is also described in [20].

- Let us first observe that, due to its very construction, each connected component C of $\mathcal{G}''$ contains at most t vertices.
- Let us now show that for any connected component C of $\mathcal{G}''$, we have $\mathcal{G}'' \setminus C \in S$. (Let us recall that $\mathcal{G}'' \setminus C$ is $\mathcal{G}''$ from which all edges incident to vertices of C have been removed.) We consider two cases.
  - Case $C = C_{m+1}$.
    We then have $\mathcal{G}'' \setminus C = \mathcal{G}'$. The fact that $\mathcal{G}' \in S$ concludes the proof of the case.

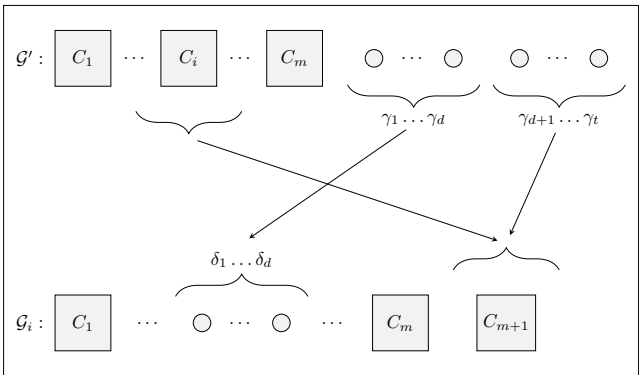

Figure 4: Using a permutation

  - Case $C = C_i$ for $1 \leq i \leq m$.
    Let $\delta_1, \dots, \delta_d$ the vertices of C_i and let $\mathcal{G}_i = \mathcal{G}'' \setminus C$. According to the permutation stability property of S there is a permutation π of the vertices of $\mathcal{G}_i$ such that $\mathcal{G} \subseteq \pi(\mathcal{G}_i)$ (Figure 4). It then follows from the fact that S is a synchrony property that $\pi(\mathcal{G}_i) \in S$ and consequently $\mathcal{G}_i \in S$, which concludes the proof of the case and the proof of the theorem. $\square_{Theorem\ 2}$

Taking the contrapositive of Theorem 1 and Theorem 2, we obtain the following corollary.

Corollary 1. *If an (n, t)-synchrony property S allows consensus to be solved, then any graph of S contains at least one connected component whose size is at least $t + 1$.*

6 A necessary and sufficient condition

This section introduces the notion of a virtual $\diamond[x+1]$bi-source and shows that the existence of a virtual $\diamond[t+1]$bi-source is a necessary and sufficient condition to solve the consensus problem in a system with message signatures and where up to t processes can commit Byzantine failures.

Definition 11. *A* $\diamond[x + 1]$bi-source *is a correct process that has an eventually synchronous bi-directional link with x correct processes (not including itself).*

From a structural point of view, a $\diamond[x + 1]$bi-source is a star made up of correct processes. (As already noticed, this definition differs from the usual one, in the sense that it considers only correct processes.)

Lemma 3. *If a graph $\mathcal{G}$ has a connected component C of size $x + 1$, a $\diamond[x + 1]$bi-source can be built inside C.*

Proof Given a graph $\mathcal{G}$ that represents the eventually synchronous bi-directional links connecting correct processes, let us assume that $\mathcal{G}$ has a connected component C such that $|C| \geq x + 1$.

A star ($\diamond[x + 1]$bi-source) can be easily built as follows. When a process p receives a message for the first time, it forwards it to all. Let us remember that, as messages are signed, a faulty process cannot corrupt the content of the messages it forwards; it can only omit to forward them. Let λ be the diameter of C and δ the eventual synchrony bound for message transfer delays. This means that, when we consider any two processes $p, q \in C$, $\lambda \times \delta$ is an eventual synchrony bound for any message communicated inside the component C. Moreover, considering any process $p \in C$, the processes of C define a star structure centered at p, and such that, for any $q \in C \setminus \{p\}$, there is a virtual eventually synchronous link (with bound $\lambda \times \delta$) that is made up of eventually synchronous links and correct processes of C, which concludes the proof of the lemma. $\square_{Lemma\ 3}$

The following definition provides us with a more general (abstract) definition of a $\diamond[x + 1]$bi-source.

Definition 12. *A communication graph $\mathcal{G}$ has a* virtual $\diamond[x + 1]$*bi-source if has a connected component C of size $x + 1$.*

Theorem 3. *An (n, t)-synchrony property S allows consensus to be solved in an asynchronous system with message authentication, despite up to t Byzantine processes, if and only if any graph of S contains a virtual $\diamond[t + 1]$bi-source.*

Proof The proof of the sufficiency side follows from the algorithm described in [12] that presents and proves correct a consensus algorithm for asynchronous systems made up of n processes where (a) up to t processes may be Byzantine, (b) messages are signed, and (c) there is a $\diamond[t+1]$bi-source[5].

When considering the necessity side, we have the following. Let S be synchrony property such that none of its graphs contains a virtual $\diamond[x+1]$bi-source. It follows from the contrapositive of Corollary 1 that S does not allow Byzantine consensus to be solved. $\square_{Theorem\ 3}$

The following corollary is an immediate consequence of the previous theorem.

Corollary 2. *The existence of a virtual $\diamond[t+1]$bi-source is a necessary and sufficient condition to solve consensus (with message authentication) in presence of up to t Byzantine processes.*

References

[1] Aguilera M.K., Delporte-Gallet C., Fauconnier H. and Toueg S., Consensus with Byzantine Failures and Little System Synchrony. *Int'l Conference on Dependable Systems and Networks (DSN'06)*, IEEE Computer Press, pp. 147-155, 2006.

[2] Attiya H. and Welch J., Distributed Computing: Fundamentals, Simulations and Advanced Topics, (2d Edition), *Wiley-Interscience*, 414 pages, 2004.

[3] Ben-Or M., Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *2nd ACM Symposium on Principles of Distributed Computing(PODC'83)*, ACM Press, pp. 27-30, 1983.

[4] Cachin Ch., Kursawe K. and Shoup V., Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, pp. 123-132, 2000.

[5] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *JACM*, 43(2):225-267, 1996.

[6] Delporte-Gallet C., Devismes S., Fauconnnier H. and Larrea M., Algorithms for Extracting Timeliness Graphs. *17th Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'10)*, Springer-Verlag LNCS #6058, pp. 127-141, 2010.

[7] Doudou A., Garbinato B., Guerraoui R. and Schiper A., Muteness failure Detectors: Specification and Implementation. *Proc. 3rd European Dependable Computing Conference (EDCC'99)*, Springer-Verlag LNCS #1667, pp. 71-87, 1999.

[8] Dwork C., Lynch N. and Stockmeyer L., Consensus in the Presence of Partial Synchrony. *JACM*, 35(2), 288-323, 1988.

[9] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *JACM*, 32(2):374-382, 1985.

[10] Friedman R., Mostéfaoui A. and Raynal M., $\diamond\mathcal{P}_{mute}$-Based Consensus for Asynchronous Byzantine Systems. *Parallel Processing Letters*, 15(1-2):162-182, 2005.

[11] Friedman R., Mostéfaoui A. and Raynal M., Simple and Efficient Oracle-Based Consensus Protocols for Asynchronous Byzantine Systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46-56, 2005.

[12] Hamouna M., Mostéfaoui A. and Trédan G., Byzantine Consensus with Few Synchronous Links. *11th Conference on Principles of Distributed Systems (OPODIS'07)*, Springer Verlag LNCS #4878, pp. 76-89, 2007.

[13] Hamouna M., Mostéfaoui A. and Trédan G., Byzantine Consensus in Signature-free Systems. *Submitted to publication.*

[14] Lamport L., Shostack R. and Pease M., The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3)-382-401, 1982.

[15] Lynch N.A., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Francisco (CA), 872 pages, 1996.

[16] Kihlstrom K.P., Moser L.E. and Melliar-Smith P.M., Byzantine Fault Detectors for Solving Consensus. *The Computer Journal*, 46(1):16-35, 2003.

[17] Okun M., Byzantine Agreement. *Springer Verlag Encyclopedia of Algorithms*, pp. 116-119, 2008.

[18] Pease M., R. Shostak R. and Lamport L., Reaching Agreement in the Presence of Faults. *JACM*, 27:228-234, 1980.

[19] Rabin M., Randomized Byzantine Generals. *24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, IEEE Computer Society Press,pp. 116-124, 1983.

[20] Raynal M., Fault-tolerant Agreement in Synchronous Message-passing Systems. *Morgan & Claypool*, 200 pages. To appear, 2010.

[21] Toueg S., Randomized Byzantine Agreement. *3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84)*, pp. 163-178, 1984.

[5] A $\diamond[t+1]$bi-source in our terminology is a $\diamond[2t]$bi-source in the parlance of [1, 12].