

Time-Free Authenticated Byzantine Consensus

Hamouma Moumen
 Computer Science Department
 University of Béjaia
 06000, Béjaia, Algeria
 Email: moumenh@gmail.com

Achour Mostefaoui
 IRISA, Université de Rennes 1
 Campus de Beaulieu,
 35042 Rennes, France
 Email: achour@irisa.fr

Abstract—This paper presents a simple protocol that solves the authenticated Byzantine Consensus problem in asynchronous distributed systems. To circumvent the FLP impossibility result in a deterministic way, synchrony assumptions should be added. In the context of Byzantine failures for systems where at most t processes may exhibit a Byzantine behavior and where not all the system is assumed eventually synchronous, Moumen et al. provide the main result. They assume at least one correct process, called $2t$ -bisource, connected with $2t$ privileged neighbors with eventually timely outgoing and incoming links. The present paper shows that a deterministic solution for the authenticated byzantine consensus problem is possible if the system model satisfies an additional assumption that does not rely on physical time but on the pattern of messages that are exchanged. The basic message exchange between processes is the query-response mechanism. To solve the Consensus problem, we assume a correct process p , called $\diamond 2t$ -winning process, and a set Q of $2t$ processes such that, eventually, for each query issued by p , any process q of Q receives a response from p among the $(n - t)$ first responses to that query. The processes in the set Q can exhibit a Byzantine behavior and this set may change over time. Whereas many time-free solutions have been designed for the consensus problem in the crash model, this is, to our knowledge, the first time-free deterministic solution to the Byzantine consensus problem.

I. INTRODUCTION

The *Consensus* problem is a fundamental paradigm to design or to implement reliable applications on top of fault prone asynchronous distributed systems. It abstracts several basic agreement problems. In the consensus problem, each process proposes a value, and the correct processes have to eventually decide on the same value initially proposed by some processes. A process is correct if it meets its specification during the whole execution, otherwise it is faulty. A faulty process can crash (*fail-stop* process) or exhibit an arbitrary behavior (*Byzantine* process). However, there is no deterministic solution for the consensus problem in asynchronous systems subject to process failures [12]. To circumvent this impossibility result, asynchronous distributed systems have to be enriched with synchrony assumptions [11], randomization [3], or unreliable failure detectors [7].

The notion of unreliable failure detectors [7] is an abstraction of synchrony assumptions in the context of crash

failures. A failure detector is a distributed oracle that provides processes with information (possibly incorrect) about process failures. Three approaches have been investigated to implement failure detectors. The first, relies on the addition of synchrony [7], called "Timer-Based". This assumption considers the partially synchronous system model [7] which is a generalization of the models proposed in [11]. In the crash-failure model, a second approach consists in adding a property that is based on the message pattern. In the latter approach, we have the query-response-based winning messages proposed in [21], [22] and the τ -model proposed in [28]. There exist also hybrid approaches [24], [25]. In a partially synchronous system model, there is a finite but unknown time (called Global Stabilization Time) after which there are bounds on process speeds and message transfer delays (but those bounds are not known).

All of the deterministic solutions proposed for the Byzantine consensus consider the partially synchronous model [2], [16], [5], [8], [9], [10], [17], [18], [20]. To solve the Byzantine Consensus problem, the notion of muteness failure detector has been proposed [9], [16]. This notion is an extension of crash failure detector to a context of mute¹ failures. Byzantine consensus algorithms proposed in [13], [14] use directly an eventually perfect muteness failure detector. Paxos-like protocols [4], [20] first look for a stable leader before solving consensus or implementing state machine replication. Finally, [10] and [18] establish lower bounds relating resiliency and (very) fast decision. [10] gives a generic algorithm that can be parametrized (w/wo authentication, fast/very fast decision) by taking into account the maximum number of processes that may crash or have malicious behavior. [6] considers an other system model based on the notion of Trusted Timely Computing Base TTCB. In this system model, the consensus protocol uses a special communication channel, called wormhole. Similarly to the works presented above, it is assumed that the wormhole allows timely communication between any pair of correct processes.

The related works cited above assume that the whole

¹Mute processes are processes that, after some time, stop sending protocol messages.

system is eventually synchronous. In this setting, a link between two processes is said to be timely at time τ if a message sent at time τ is received not later than $\tau + \delta$. The bound δ is not known and holds only after some finite but unknown time τ_{GST} (called *Global Stabilization Time*). In the context of the classical partially synchronous models [7], [11] composed of n partially synchronous processes where at most t may crash, many works [1], [15], [19] try to restrict the required synchrony property of communication to only a subset of links. The system model considered in [1] assumes at least an eventual t -source. An eventual t -source is a correct process with t outgoing eventually timely links (processes communicate using point-to-point communication primitives). On the other hand, the system model considered in [19] assumes a broadcast communication primitive and at least one correct process with t bidirectional but moving eventually timely links. These two models are not comparable [15]. In such a context, [1] proved that an $\diamond t$ -source (eventual t -source) is necessary and sufficient to solve consensus which means that it is not possible to solve consensus if the number of eventually timely links is smaller than t or if they are not outgoing links of a same correct process.

In the context of Byzantine consensus where t processes can exhibit an arbitrary behavior, Aguilera et al. [2] propose a system model with weak synchrony properties that allows solving the consensus problem. The model assumes at least an \diamond bisource (eventual bisource). An \diamond bisource is a correct process with all its outgoing and incoming links eventually timely. This means that the number of eventually timely links could be as low as $2(n - 1)$ links. Their protocol does not need authentication and consists of a series of rounds each made up of 10 communication steps and $\Omega(n^3)$ messages. More recently, Moumen et al. [23] proposed a system model that considers an eventual bisource with a scope of $2t$. The eventual bisource assumed by [2] has the maximal scope ($x = n - 1$). An eventual $2t$ -bisource ($\diamond 2t$ -bisource) is a correct process where the number of privileged neighbors is $2t$ where t is the maximum number of faulty processes. Their protocol needs authentication and consists of a series of rounds each made up of 5 communication steps and $\Omega(n^2)$ messages. When we look at the previous approaches to implement failures detectors [7], we observe that any failure detector implementation encapsulates additional assumptions to allow a deterministic solution for the consensus problem.

The second approach assumes that some pattern of messages cannot happen [21], [22], [24], [25], [26]. This property is abstracted in the notion of winning responses. The approach considers that there is a correct process p and a set Q of t processes (with p not in Q and Q can include faulty processes) such that, each time a process q in Q broadcasts a query, it receives a response from p among the first $(n - t)$ corresponding responses (such a response is

called a winning response). Note that such a property does not prevent message transfer delays from being unboundedly long as only the relative speed of messages is considered. The obtained solutions are thus much more efficient than timer-based solutions as any process only waits for the first $(n - t)$ responses (the fastest messages), whereas in the case of timer-based solutions, we wait for a delay that is the worst case message transfer delay.

The two previous approaches are not comparable. However, although the timer-based approach has been considered both in the case of crash failures (t -source) and Byzantine failures ($2t$ -bisource with authentication), the time-free approach has never been considered in the case of Byzantine faults. This paper advances the state of the art by proposing the first time-free deterministic solution to the Byzantine consensus problem. Moreover, we can see a difference with the timer-based approach. The time-free approach in the authenticated Byzantine model needs twice more winning links compared to the crash failures model whereas in the case of the timer-based approach we need four times more timely links to tolerate t Byzantine faults compared to the t links needed for crash failures. This can be explained by the query-response mechanism used by the time-free approach.

II. BASIC COMPUTATION MODEL AND CONSENSUS PROBLEM

A. Asynchronous Distributed System with Byzantine Process

We consider a message-passing system consisting of a finite set Π of n ($n > 1$) processes, namely, $\Pi = \{p_1, \dots, p_n\}$. A process executes steps (send a message, receive a message or execute local computation). The parameter t denotes the maximum number of processes that can exhibit a Byzantine behavior. A Byzantine process may behave in an arbitrary manner. It can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, send different values to different processes, perform arbitrary state transitions, etc. A correct process is one that is not Byzantine.

Processes communicate and synchronize with each other by sending and receiving messages over a network. The link from process p to process q is denoted $p \rightarrow q$. Every pair of processes is connected by two links $p \rightarrow q$ and $q \rightarrow p$. Links are assumed to be reliable. Namely, they do not create, alter, duplicate or lose messages. Such a communication network can be built atop of fair-lossy links as advocated in [15]. There is no assumption about message transfer delays and processes are not assumed to be eventually synchronous.

We assume that an authentication mechanism along with a public key infrastructure and a public key cryptography such as RSA signatures [27] are available. We assume that Byzantine processes cannot impersonate other processes. Moreover, processes sign the messages they send. Consequently, a Byzantine process cannot alter or modify a message it relays as it cannot forge the signature of the

original sending process. In our authenticated Byzantine model, we assume that Byzantine processes are not able to subvert the cryptographic primitives.

To ensure the message validity, each process has an underlying daemon that filters the messages it receives. For example, the daemon will discard all duplicate messages (necessarily sent by Byzantine processes as we assume reliable send and receive operations between correct processes). The daemon, will also discard all messages that are not syntactically correct, or that do not comply with the text of the protocol.

B. A Time-Free Assumption

Query-Response Mechanism: In this paper, we consider that each process is provided with a query-response mechanism. More specifically, any process p can broadcast a QUERY() message and then wait for corresponding RESPONSE() messages from $(n-t)$ processes (this is the maximal number of message p is sure to get without blocking). Each of these RESPONSE() messages is a winning response for that query, and the corresponding sending processes are the winning processes for that query. The responses received after the $(n-t)$ first RESPONSE() messages are the losing responses for that query and are automatically discarded. A process issues a new query only when the previous one has terminated (the first $(n-t)$ responses received). Finally, the response from a process to its own queries is assumed to always arrive among the first $(n-t)$ responses that it is waiting for. Henceforth, we reuse the definitions introduced in [21], [22], [24], [25] to define formally the notions of winning link and $\diamond x$ -winning process.

Definition 1: (Eventually winning link) Let p and q be two processes. The link $p \rightarrow q$ is eventually winning if there is a time τ such that the response from p to every query issued by q after τ is a winning response (τ is finite but unknown or will hold only after some finite time).

Definition 2: A process p is an x -winning process at time τ if:

- (1) p is correct.
- (2) There exists a set of processes X of size x , such that for any process q in X , the link $p \rightarrow q$ is eventually winning. The processes of X are said to be privileged neighbors of p .

Definition 3: A process p is an $\diamond x$ -winning process if there is a time τ such that, for all $\tau' \geq \tau$, p is an x -winning process at time τ' .

For the rest of the paper, we consider an asynchronous distributed system where the only additional assumption is the one needed by the existence of at least one $\diamond x$ -winning process with $x \geq 2t$. This means that all the links that do not participate to the establishing of the $\diamond x$ -winning property could be purely asynchronous.

C. The Consensus Problem

We consider the multivalued consensus problem, where there is no bound on the cardinality of the set of proposable values. In the multivalued consensus problem, every process p proposes a value v and all correct processes have to eventually decide on a single value among the values proposed by the processes. Formally, the consensus problem is defined by the following three properties:

- Termination: Every correct process eventually decides.
- Agreement: No two correct processes decide differently.
- Validity: If all the correct processes propose the same value v , then only v can be decided.

III. THE BYZANTINE PROTOCOL WITH AN $\diamond 2t$ -WINNING PROCESS

The proposed protocol (Figure1) uses authentication and assumes an $\diamond 2t$ -winning process. Except the coordination phase at the beginning of each round, the principle of this protocol is similar to one that has been proposed in [23]. This main difference is due to the extra assumption that strengthens the basic purely asynchronous computing model. The protocol proposed in [23] uses a timer-based assumption (it assumes an $\diamond 2t$ -bisoruce) whereas the present one uses a time-free assumption. Each process p_i executes the protocol given by Figure1. It is composed of a main task ($T1$), a decision task ($T3$) the aim of which is to allow a process to stop participating in the protocol when it decides. It implements some kind of reliable probdcast of the decision value (certified value). $T2[]$ is an array of tasks, each associated with a round r executed by process p_i . It is tasks $T2$ that implement the query-response mechanism of the coordination phase as explained in the following.

The proposed protocol uses authentication to reduce the power of Byzantine processes. Indeed, a Byzantine process p can relay falsely a value it has received from some process q . If process q signs its message and process p cannot forge the signature of q then either p relays correctly the message of q or it does not relay it at all (the signed message received by p from q is the certificate it has to append to the message it uses to relay the message it received from q). In the latter p can still lie by saying that it received no value from q . Now suppose that p has to send to all processes the majority value it has received (the most frequent value among all the values it has received). The certificate will consist of the set of signed message it has received. By this mean any process can check whether the majority value sent by the process is sound. Note that, this does not prevent some process p from cheating. For example, if a Byzantine process p receives all of the sent messages (n messages, one from each process of the system), it can build two sets of $(n-t)$ messages that lead to two different most frequent values and then sent each of these two values to different processes.

Each process p_i manages a variable est_i to store its estimate of the decision value. In order to ensure the validity property, the protocol starts with an init phase (lines 1-3) to initialize the variable est_i . This phase consists of an all-to-all message exchange that allows to initialize the local variable est_i of a process p_i to a value it has received at least $(n - 2t)$ times if any. Otherwise, est_i is set to v_i the value proposed by p_i . In the case where all correct processes propose the same value v , the only value that can be received at least $(n - 2t)$ times is v and moreover any of $(n - t)$ received messages contains at least $(n - 2t)$ times the value v . Consequently, all possible sets of $(n - t)$ received messages certify only v and no Byzantine process can introduce a wrong value as it will be discovered. If not all correct processes do propose the same value, it may happen that among the values received by a correct process p , no value is received $(n - 2t)$ times or more. In such a case, process p keeps the value it proposes and can use the set of $(n - t)$ signed messages it received as a certificate to justify why it kept its own value.

After the init phase, each process executes consecutive asynchronous rounds. Each round is composed of four communication phases and is coordinated by a predetermined process p_c (line 4).

First phase of a round r (lines 5-7). Each process that starts a round (including the coordinator of the round) first sends its own estimate (with the associated certificate) to all processes. In a separate task (line 20), Each time a process receives a valid QUERY message (perhaps from itself) containing an estimate est , it sends a RESPONSE message to the sender. If the process that responds to a query message is the coordinator of the round to which is associated the query message, the value it sends in the RESPONSE message is the coordination value. If the process that responds is not the coordinator, it responds with any value as the role of such a message is only to define winning links. as the reader can find it in line 19-22, the value sent by the coordinator is the value contained in the first valid query message of the round it coordinates. In the main task at line 6, a process p_i waits for the response from p_c (the coordinator of the round) or from $(n - t)$ responses from others processes. In the latter case, process p_i is sure that p_c is not the right winning process as its response is not winning. If a process receives a response from the coordinator then it keeps the value in a variable aux otherwise it sets aux to a default value \perp (this value cannot be proposed).

RESPONSE(r, est) messages are sent by each process from another parallel task $T2[r]$ because the coordinator of round r could be stuck in previous rounds and if it does not respond quickly, the sender on the QUERY message may receive $(n - t)$ RESPONSE messages from others processes. There is one task T per round. When the coordinator

receives the first valid QUERY message for a round it coordinates, it stores the included estimate in a local variable c_est_i . It is this value that the coordinator will send as all RESPONSE messages to the query messages associated with this round that it will receive (this allows a coordinator to coordinate a round with a certified value it has received even if it is itself lying far behind). The others RESPONSE messages sent by the others processes than the coordinator are only used to prevent processes from blocking while waiting (line 6) for the response of a faulty coordinator and the values carried by these messages are not used by processes.

If the current coordinator is a $\diamond 2t$ -winning it has at least $2t$ privileged processes among which at least t are correct processes. Consequently, at least $(t + 1)$ correct processes (the t correct processes and the coordinator itself) got the value v of the coordinator and thus set their variable aux to v ($v \neq \perp$). If the current coordinator is not a $\diamond 2t$ -winning process or if it is Byzantine, the three next phases allow correct processes to behave in a consistent way. The aim of the first phase is that if the coordinator is an $\diamond 2t$ -winning process then at least $(t+1)$ correct process will get its value at the end of line (1).

Second phase of a round r (lines 8-10). At the end of the first phase, if the current coordinator is an $\diamond 2t$ -winning process then at least $(t + 1)$ correct processes set their variable aux_i to the same non- \perp value (the value sent by the coordinator in RESPONSE messages). During the second phase, all correct processes relay, at line 9, either the value they received from the coordinator (with its certificate) or the default value \perp if they received $(n - t)$ RESPONSE messages from others processes. Each process collects $(n - t)$ valid messages and stores the values in a set V_i (line 9).

At line 9, if the coordinator is correct only one value is valid and can be relayed. Moreover, if the coordinator is a $\diamond 2t$ -winning process then any correct process p_i will get in its set V_i at least one copy of the value of the coordinator as among the $(t + 1)$ copies sent by the $(t + 1)$ correct processes that got the value of the coordinator a correct process cannot miss more than t copies (recall that a correct process collect $(n - t)$ valid messages). If the coordinator is not an $\diamond 2t$ -winning process or if it is Byzantine, this phase has no particular effect. The aim of this second phase is that if the coordinator is an $\diamond 2t$ -winning process then all the correct processes will get its value.

Third phase of a round r (lines 11-13). This phase is a filter; it ensures that at the end of this phase, at most one non- \perp value can be kept in the aux variables in the situations where the coordinator is Byzantine. If the coordinator is correct, this is already the case. When the coordinator is Byzantine two different correct processes may have set their aux_i variables to different values. In this phase, each

process collects $(n - t)$ valid messages, the values of which are stored in a set V_i . If V_i carries only the same value v ($V_i = v$) then v is kept in aux_i otherwise aux_i is set to \perp . At the end of this phase, there is at most one certified value v ($v \neq \perp$). This phase has no particular effect if the coordinator is correct. It ensures that eventhough the coordinator is Byzantine, at most one value is kept in the aux variables.

Fourth phase of a round r (lines 14-17). This phase ensures that the Agreement property will never be violated. This prevention is done in the following way. If a correct process p_i decides v during this round then if some processes progress to the next round, then v is the only certified value. In this decision phase, a process p_i collects $(n - t)$ valid messages and stores the values in V_i . If V_i carries only a unique non- \perp value v , p_i decides v . If a process p_j has received only values \perp , it is sure that no process decides during this phase and thus it can keep the value it has already stored in est_j . These two cases are exclusive. A third case may occur. If some process p_i receives both v values and \perp values, it does not know whether some process decides during this round or not. Hence, in a conservative way, it sets its variable est_i to v before starting a new round. This way, if some process decides, all other correct processes receive at least one copy of the decided value due to the intersection of the sets of received values. If no process decides, different processes may start the next round with different but still valid values.

Before deciding (line 16), a process first sends to all other processes a signed message DEC that contains the decision value (and the associated certificate). When a process p_i receives a valid DEC message at line 23, it first relays it to all other processes and then decides (not all processes decide necessarily during the same round).

IV. CORRECTNESS OF THE PROTOCOL

This section presents the proof of the termination property. Indeed, the proofs of the Agreement and Validity properties are similar to the ones presented in [23]. The proposed protocol considers at most t Byzantine processes. There are five message exchanges: lines 1-2, lines 5-6, lines 8-9, lines 11-12 and lines 14-15.

Lemma 1: If no process decides a certified value during a round $r' \leq r$, then all correct processes start round $r + 1$.

Proof: Let us first note that a correct process cannot be blocked forever in the init phase. Moreover, it cannot be blocked at line 6 because at least all correct processes respond to QUERY messages and there are at least $n - t$ correct processes.

The proof is by contradiction. Suppose that no process has decided a certified value during a round $r' \leq r$, where

r is the smallest round number in which a correct process p_i blocks forever. So, p_i is blocked at lines 9, 12 or 15.

Let us first examine the case where p_i blocks at line 9. In that case, as r is the smallest round number in which a correct process p_i blocks forever, and as line 9 is the first statement of round r where a process can block forever this means that all correct processes (they are at least $(n - t)$) eventually execute line 8. Consequently as communication is reliable between correct processes the messages sent by correct processes will eventually arrive at p_i that blocks forever at line 9. The same reasoning holds for lines 12 and 15. It follows that if p_i does not decide, it will proceed to the next round. A contradiction. ■

Theorem 1 (termination): If there is a $\diamond 2t$ -winning process in the system, then all correct processes decide eventually.

Proof: As the protocol uses authentication, if some process receives a valid DEC message, it can decide even if the message has been sent by a faulty process. Recall that a Byzantine process cannot forge a signature. If a correct process decides at line 16 or at line 23 then, due to the sending of DEC messages at line 16 or line 23, respectively, prior to the decision, any correct process will receive such a message and decide accordingly (line 23).

Now, suppose that no correct process decides. The proof is by contradiction. By assumption, there is a time τ after which there is a process p_x that is a $\diamond 2t$ -winning process. Let p_j be a correct process and one of the privileged neighbors of the winning process p_x . Let r be the first round that starts after time τ and that is coordinated by p_x . As by assumption no process decides, due to Lemma 1.

All correct processes p_i (and possibly some Byzantine processes) start round r and send a valid QUERY message to p_x (line 5). This QUERY message contains a value est which is the estimate of process p_i . When the coordinator p_x of round r receives the first QUERY message (line 20) possibly from itself, it sets a local variable c_est_x to the valid value contained in the message. Then each time process p_x receives a QUERY message related to this round (r), it sends a RESPONSE message to the sending process. If we consider any correct process p_i privileged neighbor of p_x , the RESPONSE message from p_x the coordinator to the QUERY message of p_i will be received by p_i among the first $n - t$ responses at line 6.

In the worst case, there are t Byzantine processes among the $2t + 1$ privileged neighbors of p_x (t Byzantine processes, t correct processes and itself). During the second phase, a Byzantine process can either relay the value of p_x or relay \perp arguing that it did not receive the response of p_x among the first $n - t$ RESPONSE messages (the value of p_x and \perp are the only two valid values for this round). This allows to conclude that the value v sent by p_x the coordinator of the present round is relayed at line 8) by, at least, the $t + 1$ correct

```

Function Consensus( $v_i$ )

Init:  $r_i \leftarrow 0$ ;

Task  $T1$ : % basic task %
----- init phase -----
(1) send INIT( $v_i$ ) to all;
(2) wait until ( INIT messages received from at least  $(n - t)$  distinct processes );
(3) if ( $\exists v$  : received at least  $(n - 2t)$  times ) then  $est_i \leftarrow v$  else  $est_i \leftarrow v_i$  endif;

repeat forever
(4)  $c \leftarrow (r_i \bmod n) + 1$ ;  $r_i \leftarrow r_i + 1$ ;
----- round  $r_i$  -----
(5) send QUERY( $r_i, est_i$ ) to all;
(6) wait until ( RESPONSE( $r_i, est$ ) received from  $p_c$ 
or RESPONSE( $r_i, est$ ) received from  $(n - t)$  distinct processes )
(7) if (RESPONSE( $r_i, est$ ) received from  $p_c$ ) then  $aux_i \rightarrow est$ ; % else  $\perp$  % endif;
(8) send RELAY( $r_i, aux_i$ ) to all;
(9) wait until ( RELAY( $r_i, *$ ) received from at least  $(n - t)$  distinct processes) store values in  $V_i$ ;
(10) if ( $V_i - \{\perp\} = \{v\}$ ) then  $aux_i \leftarrow v$  else  $aux_i \leftarrow \perp$  endif;
(11) send FILT1( $r_i, aux_i$ ) to all;
(12) wait until ( FILT1( $r_i, *$ ) received from at least  $(n - t)$  distinct processes) store values in  $V_i$ ;
(13) if ( $V_i = \{v\}$ ) then  $aux_i \leftarrow v$  else  $aux_i \leftarrow \perp$  endif;
(14) send FILT2( $r_i, aux_i$ ) to all;
(15) wait until ( FILT2( $r_i, *$ ) received from at least  $(n - t)$  distinct processes) store values in  $V_i$ ;
(16) case ( $V_i = \{v\}$ ) then send DEC( $v$ ) to all; return( $v$ );
(17) ( $V_i = \{v, \perp\}$ ) then  $est_i \leftarrow v$ ;
(18) endcase;
-----
end repeat

Task  $T2[r]$ : % Query-response and coordination tasks - There is one such task per round  $r$  %
(19)  $c\_est_i \rightarrow \perp$ ;
(20) upon receipt of QUERY( $r, est$ ) from  $p_j$ ;
(21) if  $p_i$  is the coordinator of round  $r$  and  $c\_est_i = \perp$  then  $c\_est_i \rightarrow est$ ;
(22) send RESPONSE( $r, c\_est_i$ ) to  $p_j$ ;

Task  $T3$ :
(23) upon receipt of DEC( $est$ ): send DEC( $est$ ) to all; return( $est$ );

```

Figure 1. A Byzantine Consensus Protocol with $\diamond 2t$ -winning

processes with which p_x has winning links (the only other possible value is \perp). Since each process collects at least $(n - t)$ RELAY messages we can conclude that all correct processes will get at least one RELAY message containing the value v of p_x and consequently will set the variable aux_i to v at line 10.

During the third phase (lines 11-13), as the value v of p_x is the only certified value (at the previous phase, any set of $n - t$ messages contain at least one value \perp), all the processes that emit a certified message emit v . This allows to conclude that all processes will have to set their aux value to v value line 13. By the same way, all processes that emit certified messages will emit v at line 14. From there we can conclude that correct processes will all decide at line 16, a contradiction. ■

V. CONCLUSION

This paper presented the first time-free deterministic protocol that solves authenticated Byzantine Consensus in an asynchronous distributed system. The protocol assumes a weak additional assumption on message pattern where at least $2t$ communication links are eventually winning. These links connect the same correct process. In favorable setting, the proposed protocol can reach decision in only 5 communication steps and needs only $\Omega(n^2)$ messages in each step. The major contribution of this paper is to show that Byzantine Consensus is possible with a weak additional assumption on the pattern of messages that are exchanged to satisfy the properties required by a $\diamond 2t$ -winning process. Compared with the time-based approach, the cost is relatively lower when we shift from crash failures to Byzantine failures.

REFERENCES

- [1] Aguilera M.K., Delporte-Gallet C., Fauconnier H. and Toueg S., Communication-efficient leader election and consensus with limited link synchrony. *Proc. 23rd ACM Symposium on Principles of Distributed Computing (PODC'04)*, ACM Press, 2004.
- [2] Aguilera M.K., Delporte-Gallet C., Fauconnier H. and Toueg S., Consensus with byzantine failures and little system synchrony. *Proc. International Conference on Dependable Systems and Networks (DSN'06)*, Philadelphia, 2006.
- [3] Ben-Or M., Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*, ACM Press, pp. 27-30, 1983.
- [4] Boichat B., Dutta P., Frolund S., and Guerraoui G., Deconstructing paxos. *SIGACT News in Distributed Computing*, **34(1)**:47-67, 2003.
- [5] Castro, M. and Liskov, B., Practical Byzantine fault tolerance. *Proc. of the 3rd Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999.
- [6] Correia M., Neves N.F., Lung L.C. and Verissimo P., Low Complexity Byzantine-Resilient Consensus. *Distributed Computing*, Volume 17, 13 pages, 2004.
- [7] Chandra T.D. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, **43(2)**:225-267, 1996.
- [8] Correia M., Neves N.F., Lung L.C. and Verissimo P., Low Complexity Byzantine-Resilient Consensus. *Distributed Computing*, Volume 17, 13 pages, 2004.
- [9] Doudou A., Garbinato B. and Guerraoui R., Encapsulating Failure Detection: from Crash to Byzantine Failures. *Proc. International Conference on Reliable Software Technologies*, Vienna (Austria), 2002.
- [10] Dutta P., Guerraoui R., and Vukolic M., Best-Case Complexity of Asynchronous Byzantine Consensus. *Technical Report EPFL/IC/200499*, EPFL, February 2005.
- [11] Dwork C., Lynch N.A. and Stockmeyer L., Consensus in the presence of partial synchrony. *Journal of the ACM*, vol 35(2):288-323, 1988.
- [12] Fischer M.J., Lynch N., and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, vol 32(2):374-382, 1985.
- [13] Friedman R., Mostefaoui A. and Raynal M., Simple and Efficient Oracle-Based Consensus Protocols for Asynchronous Byzantine Systems. *IEEE Transactions on Dependable and Secure Computing*, vol2(1):46-56, 2005.
- [14] Friedman R., Mostefaoui A., Raynal M., \diamond P-Mute-Based Consensus for Asynchronous Byzantine Systems. *Parallel Processing Letters*, vol 15(1-2):169-182, 2005.
- [15] Hutle M., Malkhi D., Schmid U., and Zhou L., Chasing the Weakest System Model for Implementing Omega and Consensus. *Research Report 74/2005*, Technische Universität Wien, Institut für Technische Informatik, July, 2006.
- [16] Kihlstrom K.P., Moser L.E., and Melliar-Smith P.M., Solving Consensus in a Byzantine Environment Using an Unreliable Fault Detector, *Proc. of the Int. Conference on Principles of Distributed Systems (OPODIS)*, pp. 61-75, 1997.
- [17] Kursawe K., Optimistic Byzantine Agreement. *Proc. of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02 Workshops)*, 2002.
- [18] Lamport, L., Lower Bounds for Asynchronous Consensus. *Distributed Computing*, vol 19(2):104-125, Springer Verlag, 2006.
- [19] Malkhi D., Oprea F., and Zhou L., Meets Paxos: Leader Election and Stability without Eventual Timely Links. *Proc. on the 19th International Conference on Distributed Computing (DISC'05)*, Cracow, Poland, pp. 26-29, 2005,
- [20] Martin J.P., and Alvisi L., Fast Byzantine Paxos. *Proc. Iof the International Conference on Dependable Systems and Networks (DSN'05)*, Yokohama, Japan, pp. 402-411, 2005.
- [21] Mostefaoui A., Mourgaya E., and Raynal M., "Asynchronous Implementation of Failure Detectors, *Proc. of the IEEE International Conference Dependable Systems and Networks (DSN '03)*, pp. 351-360, 2003.
- [22] Mostefaoui A., Mourgaya E., Raynal R. and Travers C., A Time-free Assumption to Implement Eventual Leadership. *Parallel Processing Letters*, vol 16(2):189-208, 2006.
- [23] Moumen H., Mostefaoui A., Tredan G., Byzantine Consensus with Few Synchronous Links. *Proc. of the Int. Conference on Principles of Distributed Systems (OPODIS'07)*, pp. 76-89, 2007.
- [24] Mostefaoui A., Powell D., Raynal M., A Hybrid Approach for Building Eventually Accurate Failure Detectors. *Proc. of the IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, pp. 57-65, 2004.
- [25] Mostefaoui A., Raynal M. and Travers C., Time-Free and Timer-Based Assumptions Can Be Combined to Obtain Eventual Leadership. *IEEE Transactions on Parallel and Distributed Systems*, vol. 17(7), pp. 656-666, July, 2006.
- [26] Mostfaoui A., Raynal M. and Trédan G., On the Fly Estimation of the Processes that Are Alive in an Asynchronous Message-Passing System. *IEEE Transactions on Parallel and Distributed Systems*, vol 20(6):778-787, 2009.
- [27] Rivest R., Shamir A., and Adleman L., A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, vol 21(2):120-126, February, 1978.
- [28] Widder J., and Schmid U., The Theta-Model: Achieving Synchrony without Clocks. *Distributed Computing*, vol 22(1):29-47, Springer Verlag, 2009.