International Journal of Parallel, Emergent and Distributed Systems

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/gpaa20

Minimal synchrony for implementing Timely Provable Reliable Send primitive with Byzantine failures

Mohamed Ben Othmane, Abderrahim Siam & Hamouma Moumen

To cite this article: Mohamed Ben Othmane, Abderrahim Siam & Hamouma Moumen (2023) Minimal synchrony for implementing Timely Provable Reliable Send primitive with Byzantine failures, International Journal of Parallel, Emergent and Distributed Systems, 38:4, 280-287, DOI: 10.1080/17445760.2023.2199992

To link to this article: https://doi.org/10.1080/17445760.2023.2199992

INTERNATIONAL JOURNAL OF PARALLEL, EMERGENT AND DISTRIBUTED SYSTEMS
2023, VOL. 38, NO. 4, 280–287
https://doi.org/10.1080/17445760.2023.2199992

Taylor & Francis
Taylor & Francis Group

Minimal synchrony for implementing Timely Provable Reliable Send primitive with Byzantine failures

Mohamed Ben Othmane [a,b], Abderrahim Siam [b] and Hamouma Moumen [a]

[a]Computer Science Department, University of Batna 2, Batna, Algeria; [b]University of Khenchela, Khenchela, Algeria

ABSTRACT

Broadcast abstractions are among the most important concepts in the field of fault tolerant distributed computing. These abstractions are used by consensus algorithms as a fundamental building block for ensuring that all correct processes in the system decide the same value. The Timely Provable Reliable Send primitive is among these broadcast abstractions with which we guarantee that messages are delivered correctly and in a timely manner, even in the presence of faulty processes. In this paper, we present an authenticated algorithm implementing provable reliable send primitive with very few eventually synchronous links. In other words, this algorithm assumes that there is a $\diamond\langle t+1\rangle$-sink in the system. A $\diamond\langle t+1\rangle$-sink is a correct process where the number of incoming eventually timely links that connecting it with correct processes is $(t+1)$ (including itself). We also show that a $\diamond\langle t+1\rangle$-sink is the minimal synchrony assumption for implementing this primitive in a Byzantine system where an authentication mechanism is available.

Each process p_i executes the following

(1) To *Psend* QUERY(m) to p_j;
(2) *send* QUERY(m, p_i, p_j) to all;
(3) **upon** receive QUERY(m, s, d) from p_k
(4) **if** $(s = p_k)$ and $i \neq k$ **then** *send* QUERY(m, p_k, d) to all;
(5) **if**$(p_i = d)$ and (received QUERY(m, s, d) from at least one process
and not already *Preceive* QUERY(m, s) **then** *Preceive* QUERY(m, s));
(6) **if** received QUERY(m, s, d) from $(n - t)$ distinct processes **then** *Gpoof* QUERY(m, s, d));

ARTICLE HISTORY
Received 30 September 2022
Accepted 3 April 2023

KEYWORDS
Provable reliable send primitive; Byzantine consensus; authentication; distributed system; fault-tolerance

1. Introduction

Context and motivation Since the notion of blockchain was introduced by Nakamoto [1], the problem of Byzantine consensus has attracted particular attention. The Byzantine consensus problem [2] is certainly the most important abstraction of fault-tolerant distributed computing. However, in the context of asynchronous systems, it is well known that there is no deterministic consensus algorithm even if one process may crash [3], which means that Byzantine consensus cannot be solved either-even if one process can be faulty. To make this possible, the basic asynchronous Byzantine system model has to be enriched with additional computational power. A well-known way to obtain such additional power is to add synchrony assumptions [4–6].

CONTACT Hamouma Moumen hamouma.moumen@univ-batna2.dz Computer Science Department, University of Batna 2, 53, Road of Constantine, Fesdis, Batna 05078, Algeria
This article has been corrected with minor changes. These changes do not impact the academic content of the article.

Broadcast abstractions are among the most important abstractions required to address fault-tolerant distributed computing [4,7–10]. Roughly speaking, these abstractions allow processes to disseminate information in such a way that specific provable properties concerning this dissemination are satisfied.

In the context of Byzantine Failures, broadcast abstractions are used by most consensus algorithms as a fundamental building block to disseminate messages to all processes of the system. By using the broadcast abstraction, Byzantine consensus algorithms can ensure that all processes deliver the same set of messages and that prevent Byzantine processes from sending different messages to different processes.

There are several algorithms and techniques for implementing broadcast abstractions. The implementation of broadcast abstraction depends on specific constraints related to the distributed system model. For example, some systems may use flooding-based algorithms, while others may use gossip-based protocols or other types of algorithms.

Related Works

In the context of asynchronous message-passing systems prone to Byzantine failures, there are many types of broadcast abstractions. Each of these types is defined by a set of properties, and any algorithm, that implementing it, must satisfy these properties.

In [11], Toueg introduces the one-to-all No-duplicity broadcast abstraction. This abstraction allows the processes to eliminate one bad behaviour of Byzantine processes. More precisely, a Byzantine process is prevented from sending different messages to different correct processes.

In [4], Bracha and Toueg introduce the one-to-all Byzantine reliable broadcast abstraction. This abstraction allows the correct processes to deliver the same set of messages, even the senders are faulty. In [12], Bracha proposes an optimal reliable broadcast algorithm.

In [8,9], Mostefaoui et al. introduce a powerful, yet simple, all-to-all broadcast communication abstraction suited to binary values, called BV-broadcast abstraction. This broadcast abstraction focuses on values and not on processes, which means that it does not consider the fact that 'this' value has been broadcasted by 'this' process. BV-broadcast reduces the power of Byzantine processes, in such a way that a value broadcasted only by Byzantine processes is never delivered to the correct processes.

In [13,14], Mostefaoui et al. introduce the all-to-all communication abstractions: MV-broadcast and SMV-broadcast. These abstractions extend to the 'multivalue' case the BV-broadcast and SBV-broadcast communication abstractions proposed in [9], which consider binary values only.

In contrast of the related works cited above which propose asynchronous broadcast abstractions, Aguilera et al. [7] introduce a timely broadcast abstraction, called Timely Provable Reliable Send primitive. This primitive assumes a $\diamond$-bisource in the system. Namely, the system model assumes at least one correct process with all its outgoing and incoming links eventually timely (the other links of the system are asynchronous). This means that the number of eventually timely links is $2(n-1)$. An algorithm that implementing the timely provable reliable abstraction with a $\diamond$bisource is proposed in [7]. Based on this implementation a signature-free algorithm that solving the Byzantine Consensus problem is proposed [7].

The Timely Provable Reliable Send primitive simplifies for Aguilera et al.[7] to prove that a $\diamond$bisource is a sufficient condition for the Byzantine consensus problem in asynchronous distributed systems. This primitive is used by processes to solicit the help of the coordinator of a given round. Note that, if the coordinator is not a $\diamond$bisource then it cannot give any help to correct processes that have not deciding in the algorithm presented in [7].

Contribution of the paper In this paper, we propose an authenticated algorithm implementing provable reliable send primitive with a $\diamond\langle t+1\rangle$-sink. A $\diamond\langle t+1\rangle$-sink is a correct process that having x incoming links from correct processes that are eventually timely. This means that the number of eventually timely links is $(t+1)$. In contrast of the implementation proposed in [7] that needing $2(n-1)$ eventually timely links, our implementation needs only $(t+1)$ eventually timely links. We also show that $\diamond\langle t+1\rangle$-sink is the minimal synchrony for implementing the timely provable reliable abstraction.

2. System model and synchrony assumptions

We consider a message-passing system consisting of a finite set Π of $n(n > 1)$ processes, namely, $\Pi = \{p_1, \ldots, p_n\}$. A process executes steps (send a message, receive a message or execute local computation). Value 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 does not Byzantine. A faulty process is the one that is not correct. Processes communicate and synchronize with each other by sending and receiving messages over a network. The link from process p_i to process p_j is denoted $p_i \rightarrow p_j$. Every pair of process is connected by two links $p_i \rightarrow p_j$ and $p_j \rightarrow p_i$. Links are assumed to be reliable: they do not create, alter, duplicate or lose messages.

Synchrony properties and the $\diamond\langle x\rangle$-sink Every process executes an algorithm consisting of atomic computing steps (send a message, receive a message or execute local computation). Processes are partially synchronous, in the sense that there are unknown bounds on relative speed of a correct process [5]. Hereafter, we rephrase the definition of [15] to define more formally a synchrony assumption required by our implementation of provable reliable send.

Definition 2.1: A link from a process p_i to any process p_j is timely at time τ if :

(1) no message sent by p_i at time τ is received at p_j after time $(\tau + \Delta)$ or
(2) p_j is not correct.

Definition 2.2: A process p_i is $\langle x\rangle$-sink at time τ if: p_i is a correct process and there exists a set X of correct processes (including itself) of size x, such that: for any process p_j in X, a link from p_j to p_i is timely at time τ. This means that p_i has x incoming synchronous links.

Definition 2.3: A process p_i is an $\diamond\langle x\rangle$-sink if there is a time τ such that, for all $\tau' \geq \tau$, p_j is an $\langle x\rangle$-sink at τ'.

Authentication A process may be Byzantine and disseminates a wrong value (different from the value it would have obtained if it behaved correctly). To prevent such a dissemination, the protocol uses authentication mechanisms. This implies the use of application level signatures (public key cryptography such as RSA signatures [16]). For example, process p_i has to relay a value v it has received from process p_j. Process p_j signs its message and sends it to p_i. Process p_i cannot relay v' if it cannot forge p_j's signature. Of course p_i can say that it received no value from p_j but if it relays a value from p_j, it is necessarily the value it actually received from p_j. This means that in our model we assume that Byzantine processes are not able to subvert the authentication mechanism.

Notation

The notation $\mathcal{BAMP}_{n,t}[\emptyset]$ (BAMP stands for Byzantine Asynchronous Message Passing) is used to denote the previous basic Byzantine asynchronous message-passing computation model. In the following, this model is both restricted with a constraint on t and enriched with additional assumption about synchrony. More precisely, $\mathcal{BAMP}_{n,t}[n > 3t]$ denotes the model $\mathcal{BAMP}_{n,t}[\emptyset]$ where the number of faulty processes is smaller than $n/3$, and $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \diamond\langle t+1\rangle - \text{sink}]$ denotes the model $\mathcal{BAMP}_{n,t}[n > 3t]$ enriched with an authentication mechanism such as RSA and a synchrony assumption satisfied by a $\langle t+1\rangle$-sink.

3. Provable Reliable Send primitive

Provable Reliable Send primitive is a timely broadcast abstraction introduced in [7]. It can be used by a process p_i to send a message QUERY(m) to p_j such that a third process p_r gets a proof that QUERY(m)

Each process p_i executes the following

(1) To *Psend* QUERY(m) to p_j;
(2) *send* QUERY(m, p_i, p_j) to all;
(3) **upon** receive QUERY(m, s, d) from p_k
(4) **if** $(s = p_k)$ and $i \neq k$ **then** *send* QUERY(m, p_k, d) to all;
(5) **if**$(p_i = d)$ and (received QUERY(m, s, d) from at least one process
and not already *Preceive* QUERY(m, s) **then** *Preceive* QUERY(m, s));
(6) **if** received QUERY(m, s, d) from $(n - t)$ distinct processes **then** *Gpoof* QUERY(m, s, d));

Figure 1. A Provable Reliable Send Algorithm in $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \langle t+1\rangle-\text{sink}]$).

is in transit. Provable reliable send is defined by the following three primitives:

(1) Psend QUERY(m, p_j): if a process p_i invokes Psend QUERY(m, p_j), we say that p_i *psends* QUERY(m) to p_j;
(2) Preceive QUERY(p_i, m) : if a process p_j invokes Preceive QUERY(p_i, m), we say that p_j *preceives* QUERY(m) from p_i;
(3) Gproof QUERY(m, p_i, p_j) : if a process p_r invokes Gproof QUERY(m, p_i, p_j), we say that p_r *getsproof* of QUERY(m) from p_i to p_j.

Formally, provable reliable send is defined by the following fourth properties:

- Integrity: A correct process p_j *preceives* QUERY(m) from a correct process p_i at most once, and only if p_i has previously *psent* QUERY(m) to p_j;
- Validity: If some correct process p_i *psends* QUERY(m) to some correct process p_j then eventually p_j *preceives* QUERY(m) from p_i;
- Proof-Integrity: If some correct process p_r *getsproof* of QUERY(m) from some process p_i to some correct process p_j, then p_j *preceives* QUERY(m) from p_i;
- Proof-Validity: If some correct process p_i *psends* QUERY(m) to some correct process p_j then every correct process p_r *getsproof* of QUERY(m) from p_i to p_j.

In [7], Aguilera et al. consider eventually timely provable reliable send, which guarantees that if the final receiver p_j of the message QUERY(m) is a bisource then eventually QUERY(m) cannot be received too much later than the proof. Formally, they define the following eventual timeliness propriety:

Propriety 3.1: For a system that does not need authentication, if a process p_j is a bisource, then there exists τ and T such that if some correct process p_r getsproof of QUERY(m) from some process p_i to process p_j at time τ then p_j perceives QUERY(m) from p_i by time $\max\{\tau, T\} + \Delta$.

In our paper, we consider a very weak timely propriety stated as follows:

Propriety 3.2: For a system that does not need authentication, if a process p_j is a $\langle t+1\rangle-$sink, then there exists τ and T such that if some correct process p_r getsproof of QUERY(m) from some process p_i to process p_j at time τ then p_j perceives QUERY(m) from p_i by time $\max\{\tau, T\} + \Delta$.

4. An algorithm implementing provable reliable send in $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \diamond\langle t+1\rangle - \text{sink}]$

Figure 1 presents an algorithm implementing provable reliable send primitive. It assumes that an authentication mechanism such as RSA [16] is available. A public key cryptography signatures [16] is used by a process to verify the identity of the original sender of the message and to force a Byzantine process to relay the original message that it's received if it decides to relaying it.

If a correct process p_i invokes Psend QUERY(m) to p_j, then p_i send a message QUERY(m, p_i, p_j) to all processes (lines 1–2). When p_i receives a message QUERY(m, s, d) from p_k, if p_k is the original sender ($s = p_k$) and p_i is not the original sender (this is to prevent a process p_i to send a same message an infinite times), then p_i sends QUERY(m, p_k, d) to all processes (lines 3–4). If p_i is the final destinator ($d = p_i$) of a message QUERY(m, s, d) and it receives this message from at least one process then p_i invokes Preceive QUERY(s, m) (line 5), if it has not previously invoked. If p_i has received QUERY(m, s, d) from $(n - t)$ distinct processes, then p_i invokes Gproof QUERY(m, s, d) (line 6).

5. Proof of the algorithm

Lemma 5.1 (Integrity): *A correct process p_j Preceive QUERY(p_i, m) from a correct process p_i at most one time, and only if p_i has previously Psend QUERY(m) to p_j.*

Proof: If a correct process p_j Preceive QUERY(p_i, m) then due to the invocation of the Psend primitive to send QUERY(m) message at line 2, p_j can't Preceive QUERY(p_i, m) from a correct process p_i more than once due to always verification before its invocation of Preceive primitive at line 5. The proof is by contradiction. Suppose that p_i invokes Psend Q(m) to p_j and p_j is not previously already Preceive QUERY(p_i, m) from p_i. So, p_j will never invoke Preceive QUERY(p_i, m) from p_i. This means that p_j will receive QUERY(m, p_i, p_j) message from less than one process (not receive it). When p_i invokes Psend QUERY(m) to p_j, then it sends QUERY(m, p_i, p_j) to all processes and any correct process relays this message to all processes when its reception. Consequently as communication is reliable between correct processes the messages sent by correct processes will eventually arrive and p_j will receive more than One QUERY(m, p_i, p_j) messages from at least one process, such that p_j is not previously already Preceive QUERY(p_i, m) from p_i. Consequently, p_j invokes Preceive QUERY(p_i, m) from p_i. A contradiction. ■

Lemma 5.2 (Validity): *If some correct process p_i Psend QUERY(m) to some process p_j then eventually p_j Preceive QUERY(p_i, m) from p_i.*

Proof: Let us first note that, if a correct process p_i invokes Psend QUERY(m) to a correct process p_j then it sends QUERY(m, p_i, p_j) to all processes and each correct process, with the exception of the original sender, will sent this message to all processes when its reception. So, suppose that p_i invokes Psend QUERY(m) to a correct process p_j. Consequently, any correct process sends QUERY(m, p_i, p_j) to p_j. As links between processes are reliable and there are at least $(n - t)$ correct processes in the system, at least One message sent by a correct process will eventually received by p_j. Hence, p_j will eventually invoke Preceive QUERY(p_i, m) from p_i. ■

Lemma 5.3 (Proof-Integrity): *If some correct process p_k invokes Gproof of QUERY(m, s, d) from some process p_i to some correct process p_j then p_j invokes Preceive QUERY(p_i, m) from p_i.*

Proof: If a correct process p_k invokes Gproof of QUERY(m, s, d) message from p_i to p_j then p_k receives this message from at a least $(n - t)$ distinct processes. So, suppose that a correct process p_j will never invoke Preceive QUERY(p_i, m) from p_i. The proof is by contradiction. By hypothesis, p_k invokes Gproof of QUERY(m, s, d) from p_i to p_j. This means that p_k receives QUERY(m, s, d) from at least $(n - t)$ distinct processes, where at least $(n - 2t)$ are corrects, because links are reliable and there are at most t Byzantine processes. This implies that at least $(n - 2t) \geq 1$ correct processes sent QUERY(m, s, d) message to p_j and p_j will invoke Preceive QUERY(p_i, m) from p_i. A contradiction. ■

Lemma 5.4 (Proof-Validity): *If some correct process p_i invokes Psend QUERY(m) to some process p_j then every correct process p_k invokes Gproof of QUERY(m, s, d) from p_i to p_j.*

Proof: Suppose that a correct p_k will never invoke Gproof of QUERY(m, s, d) from p_i to p_j. The proof is by contradiction. By hypothesis, a correct p_i invokes Psend QUERY(m) to a correct process p_j. Then p_i sends QUERY(m, pi, pj) to all processes and each correct process, with the exception of the original sender, will sent this message to all processes including p_k when its reception. Consequently, as links are reliable and there are at least $(n - t)$ correct processes in the system, the messages sent by correct processes will be received by p_k and it invoke Gproof of QUERY(m, s, d) from p_i to p_j. A contradiction. ■

Lemma 5.5 (Eventually Timely Provable Reliable Send): *In the system model* $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \langle t+1\rangle\text{−sink}]$*, if* p_i *is a* $\langle t+1\rangle$*-sink, then there exists* τ *and T such that if some correct process* p_r *getsproof of QUERY(m) from some process* p_i *to process* p_j *at time* τ*, then* p_j *perceives QUERY(m) from* p_i *by time* $\max\{\tau, T\} + \Delta$.

Proof: Suppose that p_j invokes Preceive QUERY(p_i, m) from p_i at time $\tau' > \max\{\tau, T\} + \Delta$. The proof is by contradiction. By hypothesis, p_j is an $\langle t+1\rangle$-sink at a time T and a correct process p_k invokes Gproof of QUERY(m, p_i, p_j) from p_i to a correct process p_j at a time τ. A process p_i is a $\diamond\langle t+1\rangle$-sink, this means that it have $(t + 1)$ incoming timely links. If a correct process p_i invokes Psend QUERY(m) to a correct process p_j then p_i sends QUERY(m, p_i, p_j) to all processes and any correct process relays this message to all processes. This implies that all correct processes will invoke Gproof of QUERY(m, p_i, p_j) from p_i to p_j at a time τ.

To simplify the proof, we assume that all correct process invokes Gproof of QUERY(m, p_i, p_j) at the same time τ. This means that any correct process p_k receives from at a least $(n - t) \geq 2t + 1$ distinct processes. In the worst case, there are t QUERY(m, p_i, p_j) messages sent to p_k by Byzantine processes and the rest of $(n - 2t) \geq t + 1$ messages are sent by correct processes at a time τ and at the same time those correct processes sent QUERY(m, p_i, p_j) message to p_j. A process p_j is connected with $(t + 1)$ processes by incoming eventually timely links. When we consider that among the $(n - t)$ QUERY(m, p_i, p_j) messages received by p_k there are t messages sent by Byzantine processes and the t QUERY(m, p_i, p_j) missing message are from t correct processes that having eventually timely links with p_j. This means that p_j will receive QUERY(m, p_i, p_j) from one correct process. Therefore, p_j receives *One* QUERY(m, p_i, p_j) from a correct process at the time $\max\{\tau, T\} + \Delta$ and p_j invokes Preceive QUERY(p_i, m) from p_i at the same time. A contradiction. ■

Theorem 5.1: *The algorithm described in Figure 1 implements provable reliable send primitive in the system model* $\mathcal{BAMP}_{n,t}[n > 3t, RSA]$.

Proof: The proof follows directly from Lemmas 5.1, 5.2, 5.3 and 5.4. ■

Theorem 5.2: *The algorithm described in Figure 1 implements eventually Timely Provable Reliable Send primitive in the system model* $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \langle t+1\rangle\text{−sink}]$.

Proof: The proof follows directly from Theorem 5.1 and Lemma 5.5. ■

6. Impossibility results

The proof of the sufficiency of having $\diamond\langle t+1\rangle$−sink follows from the algorithm described in Section 4. We have showed that, in the system model $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \diamond\langle t+1\rangle\text{−sink}]$, a $\diamond\langle t+1\rangle$−sink is the sufficient synchrony for implementing eventually Timely Provable Reliable Send primitive.

In this section, we show that $\diamond\langle t+1\rangle$−sink is the necessary synchrony for implementing eventually Timely Provable Reliable Send primitive in the system model $\mathcal{BAMP}_{n,t}[n > 3t, RSA,]$. Consequently, we show that $\diamond\langle t\rangle$−sink is not sufficient for implementing eventually Timely Provable Reliable Send primitive.

Let us note that a correct process p_i cannot invoke Preceive of any valid message (a massage with correct signatures), with timely manner, if not receives this message more than one time from correct process that having timely links with p_i. It is natural that if the final receiver does not receive one valid copy of a message from a process with a timely way, then eventually Timely Provable Reliable Send primitive cannot be implemented.

To make our proof very easy and simple to understand, we examine the case where there is only one process may be Byzantine.

The case $t = 1$ To show that it is impossible to implement provable eventually timely reliable send, even if there is only one process may be faulty in the system, we consider that there is a $\langle 1 \rangle$−sink. In this case, all links of the system are asynchronous, but all processes are partially synchronous according the system model defined in [5].

Theorem 6.1: *Eventually Timely Provable Reliable Send primitive can't be implemented in the system model* $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \langle 1 \rangle\text{−sink}]$.

Proof: The proof is by contradiction. Assume that there is an algorithm A implementing eventually Timely Provable Reliable Send that tolerating one failure ($t = 1$). Construct a run R as follows. Choose some process p_j to be a $\langle 1 \rangle$−sink at a time T. Choose some correct process $p_i \neq p_j$, and let use A to send a message m to p_j. Choose a third correct process $p_k \notin \{p_i, p_j\}$ that getsproof of m from p_i to p_j at a some time τ, by choosing some Δ and making every message psent by p_i be preceived by time $t_1 = Max\{\tau, T\} + \Delta$. Pick some correct process $p_r \notin \{p_i, p_j, p_k\}$ that having an asynchronous link with p_j, because there are no timely incoming links for p_j. This means that, at a time τ', every message sent by p_r or p_i to p_j is received within time $t_2 \leq t_1$. Consequently, if p_k getsproof of the message m, psent by p_i to p_j at time τ from all processes except of p_z (assumed as crashed), then p_j receives m from itself at time t_2 and preceive m at the same time $t_2 \leq t_1$.

Now consider a run R' that is very similar to R, but at time τ', p_j does not send m to itself and it send it after time τ''. In R', p_k getsproof from all processes except of p_j, because p_z behaves correctly in this run. For process p_k, runs R and R' are indistinguishable. This means that p_j receives m from itself at time $t_3 > t_1$. Consequently, p_j preceives m at time t_3 after t_1. A contradiction with the timeliness propriety of A. ■

7. Conclusion

In this paper, we presented an algorithm that implementing eventually Timely Provable Reliable Send primitive in the system model $\mathcal{BAMP}_{n,t}[n > 3t, RSA, \langle t+1 \rangle\text{−sink}]$. Our implementation guarantees that a message m sent by a correct sender p_i will arrive with timely way to the receiver p_j if a third party p_r getsproof of m from the original sender to the receiver if p_j is a $\langle t+1 \rangle$−sink. This will occur even the link between the sender and the receiver is asynchronous.

For an Authenticated Byzantine asynchronous distributed system where $n \geq (3t + 1)$, we have shown that $\langle t+1 \rangle$−sink is the minimal synchrony for implementing eventually Timely Provable Reliable Send primitive. In other words, it is impossible to implement eventually Timely Provable Reliable Send primitive with only $\langle t \rangle$−sink.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Mohamed Ben Othmane http://orcid.org/0000-0002-3690-2394
Abderrahim Siam http://orcid.org/0009-0006-7578-768X
Hamouma Moumen http://orcid.org/0000-0002-1986-7590

References

[1] Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. [Online]. https://bitcoin.org/bitcoin.pdf.
[2] Pease M, Shostak R, Lamport L. Reaching agreement in the presence of faults. J ACM. 1980;27:228–234.
[3] Fischer MJ, Lynch N, Paterson MS. Impossibility of distributed consensus with one faulty process. J ACM. 1985;32(2):374–382.
[4] Bracha G, Toueg S. Asynchronous consensus and broadcast protocols. J ACM. 1985 Oct;32(4):824840.
[5] Dwork C, Lynch NA, Stockmeyer L. Consensus in the presence of partial synchrony. J ACM. 1988;35(2):288–323.
[6] Martin J-P, Alvisi L. Fast Byzantine consensus. IEEE Trans Dependable Secure Comput. 2006;3(3):202–215.
[7] Aguilera MK, Delporte-Gallet C, Fauconnier H, et al. Consensus with Byzantine failures and little system synchrony. In: Proceedings International Conference on Dependable Systems and Networks (DSN'06); Philadelphia. IEEE; 2006.
[8] Mostefaoui A, Moumen H, Raynal M. Signature-free asynchronous byzantine consensus with $t < n/3$ and $o(n2)$ messages. In: Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC 2014); Paris, France. ACM Press; 2014.
[9] Mostefaoui A, Moumen H, Raynal M. Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $o(n2)$ messages, and $O(1)$ expected time. J ACM. 2015;62(4):1–21.
[10] Srikanth TK, Toueg S. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distrib Comput. 1987;2(2):80–94.
[11] Toueg SRandomized Byzantine agreement. In: Proceedings 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84); Vancouver, British Columbia, Canada. ACM Press; 1984. p. 163–178.
[12] Bracha G. Asynchronous Byzantine agreement protocols. Inf Comput. 1987;75(2):130–143.
[13] Mostéfaoui A, Moumen H, Raynal M. Modular randomized Byzantine k-set agreement in asynchronous message-passing systems. In: Proceedings 17th Int'l Conference on Distributed Computing and Networking (ICDCN'16); Singapore. ACM Press; 2016. 10 pp.
[14] Mostéfaoui A, Moumen H, Raynal M. Randomized k-set agreement in crash-prone and Byzantine asynchronous systems. Theor Comput Sci. 2018;709:80–97.
[15] Hutle M, Malkhi D, Schmid U, et al. Chasing the weakest system model for implementing. Research Report 74/2005, Technische Universität Wien, Institut für Technische Informatik; 2006 Jul; Wien, Austria.
[16] Rivest RL, Shamir A, Adleman LM. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM. 1978 Feb;21(2):120–126.