Reliable Broadcast in Anonymous Distributed Systems with Fair Lossy Channels

Jian Tang , Mikel Larrea , Sergio Arévalo , and Ernesto Jiménez

Abstract—**Reliable Broadcast (RB) is a basic abstraction in distributed systems, because it allows processes to communicate consistently and reliably to each other. It guarantees that all correct process reliable deliver the same set of messages. This abstraction has been extensively investigated in distributed systems where all processes have different identifiers, and the communication channels are reliable. However, more and more anonymous systems appear due to the motivation of privacy. It is significant to extend RB into anonymous system model where each process has no identifier. In another hand, the requirement of reliable communication channels is not always satisfied in real systems. Hence, this paper is aimed to study RB abstraction in anonymous distributed systems with fair lossy communication channels.**

In distributed systems, symmetry always mean that two systems should be considered symmetric if they behave identically, and two components of a system should be considered symmetric if they are indistinguishable. Hence, the anonymous distributed systems is symmetry. The design difficulty of RB algorithm lies in how to break the symmetry of the system. In this paper, we propose to use a random function to break it. Firstly, a non-quiescent RB algorithm tolerating an arbitrary number of crashed processes is given. Then, we introduce an anonymous perfect failure detector AP^*. Finally, we propose an extended and quiescent RB algorithm using AP^*.

Index Terms—**Anonymous distributed system, asynchronous system, reliable broadcast, fair lossy communication channels, failure detector, quiescent.**

I. Introduction

Reliable Broadcast (*RB*) is a fundamental service in distributed systems that helps to build reliable distributed applications. It is used to disseminate messages among a set of processes with *RB-broadcast()* and *RB-deliver()* operations, which was introduced in [1]. In short, RB is a broadcast service that requires all non-crashed processes deliver the same set of messages, and that all messages sent by non-crashed processes must be delivered by all non-crashed processes.

This service as has been extensively studied in classic distributed systems, i.e., in which each process has a unique identifier ([2], [3], [4]). On the other hand, this study in anonymous distributed systems, i.e., processes have no identifiers and are programmed identically [5], has few result. In [6], the RB abstraction has been studied in anonymous distributed systems with reliable communication channels. Anonymity is a new and challenging point in distributed computing. In classic message-passing distributed systems, processes communicate with each other by passing messages. Because they all have unique identifiers, senders can choose the recipients of their messages, and recipients are aware of the identities of the senders of messages they receive [7]. However, all these rules have to be changed in anonymous distributed systems. In anonymous systems, when a process receives a message, it can not distinguish this message comes from which sender. The difficulty in the design of any distributed algorithms in anonymous systems lies in how to break the symmetry of system, i.e., how to distinguish messages from the same process or different processes. In the literature, there are three main methods used to break symmetry in anonymous distributed systems: randomization [8], leader election [9], and direction sensitive [10]. Informally, randomization means that there is a random function subject to a distribution which is used to give random name to each process; leader election is a deterministic form of symmetry breaking that an elected leader can assign names to processes, count the number of processes of the system, etc.; direction sensitive refers to that each process has local port number, it senses the message received or sent from/to which port.

The development of anonymous distributed systems is very quick. In general, this trend is caused by two important reasons: privacy and practical constraints. In some distributed applications, like peer-to-peer file systems, users do not want to be identified [11]. Other applications that use sensor networks has constraints in where a unique identity is not possible

to be included in each sensor node (due, for example, to small storage capacity, reduced computational capacity, or a huge number of elements to be identified) [12]. As we have known, the first paper studied about anonymous systems was addressed by D. Angluin [13]. Then, lots of paper appeared in this field, as ring anonymous networks, and shared memory anonymous systems ([14], [15], [16], [17]).

Moreover, the study of RB (no matter in classic or anonymous distributed systems) usually assume that the communication channels are reliable (if a process p sends a message to a process q, and both p and q are correct, then q eventually receives m). However, most of the communication channels in real systems are unreliable (e.g., fair lossy, which means that if a message is sent an arbitrary but finite number of times, then there is no guarantee on its reception, because the channel can lose an infinite number of messages [18]). In this regard, several works have addressed the issue of how to construct reliable channels over unreliable channels in classic distributed systems [18], [19]. To the best of our knowledge, RB has not been studied in anonymous distributed systems with unreliable channels.

Our Contributions This is paper is devoted to the implementation of Reliable Broadcast abstraction in anonymous asynchronous message-passing systems that processes are crash prone and communication channels are fair lossy. We have two main contributions:

- The paper proves that Reliable Broadcast abstraction can be implemented in anonymous asynchronous system with fair lossy communication channels and any number of correct processes. Two implementation algorithms (non-quiescent and quiescent) and corresponding proofs are given in this paper. Besides, in these algorithms, every process is not necessary to know the total number of processes in the systems.
- A new class of anonymous failure detector AP^* is proposed. This failure detector outputs a set of paris of $label$ and $number$, where the $label$ represents a temporal identifier of a process (ID is used in the failure detector layer) and the $number$ represents the number of processes who have known this $label$ (in the failure detector layer). The information of correct processes from this AP^* is used to make the Reliable Broadcast algorithm to be quiescent.

Roadmap This paper is organized as follows. The system model and several definitions are presented in Section 2. One non-quiescent algorithm is proposed in section 3 to implement *Reliable Broadcast* abstraction in the model of anonymous asynchronous systems with fair lossy channels. In Section 4, a new class of failure detector AP^* is defined firstly, then a quiescent Reliable Broadcast implementation algorithm with AP^* is given. Finally, this paper is concluded by the Section 5.

II. System Model and Definitions

In this paper, an anonymous asynchronous system is considered as a system in which processes have no identifiers and communicate with each other via fair lossy communication channels. Two primitives are used in this system to send and receive messages: *broadcast()* and *receive()*. We say that a process p_i broadcasts a message m to all processes (including itself) when it invokes $broadcast_i(m)$; a process p_i receives a message m when it invokes $receive_i(m)$. Note that in an anonymous distributed system, when a process receives a message m it cannot determine who is the sender of m. Finally, we assume that there is a global clock whose values are natural numbers, but processes cannot access it.

Process All processes are anonymous, that means they have no identifiers and execute the same algorithm. Furthermore, all processes are asynchronous, that is, there is no assumption on their respective speeds. In this paper, the anonymous distributed system is constituted by a set of n anonymous processes, denoted as $\Pi = \{p_i\}_{i=1,\ldots,n}$, such that its size is $|\Pi| = n$. We consider that $i(1 \leq i \leq n)$ is the index of each process of the system. This index cannot be known by processes, it is just used as a notation to simplify the description of the algorithm.

There is a global clock T whose values are positive natural numbers. Note that T is an auxiliary concept that we only use it for notation, but processes cannot check or modify it.

Failure model A process stops to execute the algorithm any more is crashed. A process that does not crash in a run is $correct$ in that run, otherwise it is $faulty$. We use $Correct$ to denote the set of correct processes in a run, and $Faulty$ to denote the set of faulty processes. A process executes its algorithm correctly until it crashes. A crashed process can not execute any more statements or recover.

Communication Each pair of processes are connected by bidirectional fair lossy communication channels. A communication channel between two processes p and q is called as fair lossy communication channel if it satisfies the following properties [20]:

- Fairness: If p sends a message m to q an infinite number of times and q is correct, then q eventually receives m from p.
- Uniform Integrity: If q receives a message m from p, then p previously sent m to q; and if q receives m infinitely often from p, then p sends m infinitely often to q.

Processes communicate among them by sending and receiving messages through these channels. We assume that these channels neither duplicate nor create messages, but may lost messages.

Reliable Broadcast Reliable Broadcast is one type of fault-tolerant broadcast service in distributed systems. It requires that all correct processes deliver the same set of messages, and that all messages sent by correct processes must be delivered to all correct processes. Formally, reliable broadcast is defined by two primitives: $RB\_broadcast(m)$ and $RB\_deliver(m)$. They satisfy three properties as follows:

- *Validity*: If a correct process broadcasts a message m, then it eventually delivers m.

- *Agreement*: If a correct process delivers a message m, then all correct processes eventually deliver m.
- *Uniform Integrity*: For any message m, every process delivers m at most once, and only if m was previously broadcast by a process.

Note that Validity and Agreement imply that all correct processes deliver all the messages broadcast by correct processes.

Failure Detector The failure detector is a module that provides each process a read-only local variable containing failure information (may be unreliable) of processes. The notion of failure detector is proposed and developed by Chandra and Toueg in their seminal paper [21]. The failure detector is defined by the completeness and accuracy properties. A failure detector history H is a function from $\Pi \times \mathcal{T}$ to 2^{Π}. $H(p, t)$ is the value of the failure detector module of process p at time t. If $q \in H(p, t)$, it means that p suspects q at time t in H.

Each process has access to its local failure detector for obtaining failure information of processes. They can be divided into different classes according to the quality of information they give. A failure detector can make mistakes by wrongly suspect a running process as a crashed one or does not suspect a really crashed process. Hence, the failure detector may repeatedly trust or suspect one process. This character of failure detector implies that any two failure detector of different processes may provide different failure information.

Notation The system model is denoted by either $AAS\_F_{n,t}[\emptyset]$ or $AAS\_F_{n,t}[D]$. AAS_F is an acronym for anonymous asynchronous message passing distributed system with fair lossy communication channels; $\emptyset$ means that there is no additional assumption, while D means that the system is enriched with a failure detector of class D. The variable n represents the total number of processes in the system, and t represents the maximum number of processes that can crash.

III. Implementing Reliable Broadcast in $AAS\_F_{n,t}[\emptyset]$

In this section, we present an algorithm implementing $Reliable\ Broadcast$ abstraction in anonymous asynchronous systems with fair lossy communication channels. This algorithm can run independently of the number of faulty processes.

In anonymous systems, processes have no identifiers making the design of Reliable Broadcast algorithm very difficult. In order to solve this difficulty, we summarize the main challenge firstly. It is well known that each message can be identified by both the identifier of its sender and a sequence number in classic systems. However, in anonymous systems, it is impossible to use the identifier of process (because processes do not have identifiers) or to distinguish all identical messages only by a sequence number (because different process may use the same one). If a process receives a message, it does not know where it comes from. The obvious idea, like most works in the literature, is to assign an identifier to each process, and then run the algorithm of eponymous distributed systems. In fact, this method has broken the anonymity of the system that a process can be tracked by its message flow. Because a fixed identifier is attached to all messages RB-broadcast by one process. The possibility of successful tracking is elevated by the Big Data and Cloud Computing technologies. This has been confirmed by the state of the art research result of MIT [22]. In another words, the anonymity gained by the way of hiding the identifiers of processes is not real anonymity. Then, we give a definition of anonymity:

In distributed systems, anonymity means that processes have no identifier, and also means that the relationship between messages and their senders are unknown and untrackable.

According to this definition, the system is not really anonymous if identifiers are assigned to processes that the relationships between message and its sender can be tracked.

In fact, to handle the design difficulty of Reliable Broadcast algorithm in anonymous systems without breaking the anonymity, we do not necessarily need the identifiers of processes or assign identifiers to them. Instead, what we really need is the capability to make every message in the system to be unique (break the symmetry of systems). In this paper, we propose that each process manages a random function to assign a unique one-time label to each message. When one process reliable broadcasts a message, the local random function of this process generates a random number which will be piggybacked as a label (denoted by tag) to this message. Note that, this unique label will neither be used as an address for sending messages nor to identify a certain process. Moreover, it is assumed that neither one random function nor two can generate identical label assigned to two different messages.

Though the probability of assigning a unique label to each message is very high, this assignment does not break the anonymity of the system that no process knows the mapping relationship between a tag and a process. Moreover, according the result of [10] and [23], a simple probabilistic analysis using a well known "birthday paradox" shows that the probability of a collision is nearly zero if 100 concurrent processes in a very large-scale system draw from a 128-bit field. Following this result, processes RB-broadcast messages with an identical label in one instance is really low. Moreover, in order to avoid the collision in different instances, each process has a variable to record the last sequence number of the message broadcast by itself. With these two parts, a process first draw a random number, then piggyback a sequence number obtained by increasing the last sequence number by 1.

Figure 1 presents the algorithm in detail, each process owns a random function $random()$ which is used to assign a unique tag to every message before to broadcast it. In order to facilitate the description, let's consider a process p_i (index i is used just for description, no process knows which process is p_i, even itself).

Description of the algorithm:

Every process p_i manages two local sets:

- MSG_i, which records all messages either broadcast or received by p_i.
- $RB\_DELIVERED_i$, which records all messages reli-

```
1 Initialization
2    sets MSG_i, RB_DELIVERED_i empty
3    activate Task 1

4 When RB_broadcast_i(m) is executed
5    tag ← random_i()
6    insert (m, tag) into MSG_i

7 When receive_i(MSG, m, tag) is executed
8    if (m, tag) is not in MSG_i then
9       insert (m, tag) into MSG_i
10   end if
11   if (m, tag) is not in RB_DELIVERED_i then
12      insert (m, tag) into RB_DELIVERED_i
13      RB_deliver_i(m)
14   end if

   Task 1:
15 repeat forever
16   for every message (m, tag) in MSG_i do
17      broadcast_i(MSG, m, tag)
18   end for
19 end repeat
```

Figure 1. Reliable Broadcast in $AAS\_F_{n,t}[\emptyset]$ (code of p_i)

ably delivered by p_i.

The algorithm works as follows:

Initially, MSG_i and $RB\_DELIVERED_i$ are set to empty, and Task 1 is activated (lines 1-3). When p_i calls $RB\_broadcast_i(m)$ (line 4), its $random_i()$ generates a random tag for m firstly (line 5). Then, p_i inserts (m, tag) into MSG_i (line 6), so that m will be broadcast periodically to all processes in Task 1 (lines 15-19).

When $receive_i(MSG, m, tag)$ is executed (line 7), p_i inserts (m, tag) into MSG_i if this is the first reception of m (lines 8-10). Then, p_i checks whether m has already been delivered or not (line 11). If not, p_i inserts m into $RB\_DELIVERED_i$ and then reliably delivers it (lines 12-13).

In Task 1, every message in MSG_i is periodically broadcast by p_i in order to overcome the message losses caused by the fair lossy communication channels.

Correctness Proof:

Lemma 1: *If a correct process $RB\_broadcast$ a message m, then it eventually $RB\_deliver$ m. (Validity)*

Proof: Let us consider a non-fault process p_i (i is used for description, no process knows which process is p_i) that invokes $RB\_broadcast(m)$. It firstly generates a unique random number as a tag to this message m (Line 5), then inserts (m, tag) into its set MSG_i to $broadcast$ it to all processes (included itself) (Lines 6, 15-19). For p_i is correct, this Task 1 will execute forever to disseminate m (broadcast infinite times). Then, together with the fairness property of fair lossy communication channels, p_i will receive m eventually (by itself). Because this is the first reception of m and m has not been $RB\_delivered$ before by p_i, p_i $RB\_deliver()$ m one time(Lines 11-14). We finish the proof of this Lemma 1. ■

Lemma 2: *If a correct process $RB\_deliver$ a message m, then all correct process eventually $RB\_deliver$ m. (Agreement)*

Proof: Let us assume, by the way of contradiction, that the claim is not true. It means that if a correct process p_i has delivered a message m, then, eventually there exists at least one correct process does not deliver it.

We suppose that p_i has $Rb\_delivered$ m. According to the algorithm (Line 6), m must be inserted into the set MSG_i by p_i when $RB\_broadcast_i(m)$ is called before $RB\_deliver()$ it. And p_i is a correct process, it executes Task 1 forever to broadcast every message (including (m, tag)) that existed in its set MSG_i to all processes (Lines 15-19). According to the property of fair lossy communication channel, if a message is broadcast an infinite times by a correct process, this message must be received by one correct process eventually. If the assumption is correct that there exists one correct process does not deliver m which means that this correct process does not receive m, then we get a contradiction here. Hence, the assumption is incorrect, and we complete the proof of Lemma 2. ■

Lemma 3: *For any message m, every correct process $RB\_deliver$ m at most once, and only if m was previously $RB\_broadcast$ by $sender(m)$. (Integrity)*

Proof: The second part of this lemma that any message m was previously $RB\_broadcast$ by its sender is trivial, due to the fact that each process only forward messages it has received and fair lossy channels do not create, duplicate, or garble messages.

Then, we focus on the proof of the first part of this lemma. It is supposed that each message has a unique tag, and together with that each process has a set $RB\_DELIVERED_i$ to record all messages that have delivered (Line 12). Even though each message can be broadcast forever by correct processes and will be received by every correct process for infinite times (Lines 15-19), every message has to be checked whether it has already been $RB\_delivered$ when it is received by a correct process. So, the set $RB\_DELIVERED_i$ guarantees that no message m will be $RB\_delivered$ more than once. We finish the proof of Lemma 3. ■

Theorem 1 *Algorithm 1 guarantees the property of reliable broadcast.*

Proof: According to Lemma 1, 2 and 3, it is trivial to see that Theorem 1 is correct. ■

IV. Implementing Reliable Broadcast Quiescently in $AAS\_F_{n,t}[AP^*]$

Observe that the algorithm of Figure 1 is non-quiescent due to the permanent periodical broadcast of received messages in Task 1. Hence, in this section we address the design of a quiescent algorithm implementing Reliable Broadcast. The approach followed consists in eventually deleting every message from the set MSG. According to the properties of Reliable Broadcast, the periodical broadcast of Task 1 could be safely terminated when all the messages $RB\_delivered$ by any correct process have been received by all correct processes. In other words, we could delete a message from the set MSG when it has been received by all correct processes.

Based on the previous, the design of a quiescent algorithm is reduced to the following two sub-problems: (i) determining the set of all correct processes in the system, and (ii) confirming that a message has been received by all processes in this set. We will address the first sub-problem with a failure detector, and then use it to solve the second sub-problem algorithmically.

A. Failure Detector AP^*

The failure detector abstraction, proposed by Chandra and Toueg [21], provides (possibly unreliable) failure information of processes. It is defined by both completeness and accuracy properties. In non-anonymous distributed systems, the failure information is usually composed of the identifiers of processes. However, in anonymous distributed systems processes have no identifiers. Hence, the main difficulty in defining a failure detector for anonymous distributed systems is how to give meaningful failure information about processes without identifiers. In this regard, we follow the approach of the failure detector $A\Sigma$, introduced by Bonnet and Raynal [24], that assigns a random $label$ to each process as a temporal identifier. This assignment neither break the anonymity of systems nor release the information of the relationship between a message and its sender. Because failure detector is a separate modular and the assignment is deployed inside this modular. In other words, the mapping relationship of a process and a label is packed inside of the failure detector forever.

As mentioned before, a process p_i can delete a message m from its MSG_i when it has received acknowledge messages to this m from all correct processes. So, the failure detector has to output the information of all correct processes. It means that this failure detector must have a strong completeness property that eventually correct processes do not trust any process that crashes[1], and a strong accuracy property that a process cannot be trusted once it is crashed (may be need a little time).

We define a perfect anonymous failure detector AP^* (the anonymous counterpart of Chandra-Toueg's perfect failure detector P) that satisfies strong completeness (eventually all correct processes are permanently trusted by every correct process) and strong accuracy (eventually correct processes do not trust any process that has crashed). AP^* provides each process p_i with a read-only local variable $a\_p_i^*$ that contains pairs ($label$, $number$), where $label$ is a temporal identifier of a process and $number$ is the number of correct processes who have known $label$. For example, if process p_j's local variable $a\_p_j^*$ contains $\{(label_1, number_1), ..., (label_i, number_i), ..., (label_n, number_n)\}$, it means that p_j has known the $labels$ of n processes and the corresponding number of correct processes who have known each $label$. The definition of AP^* is as follows:

- AP^*-$completeness$: There is a time after which a_p^* permanently contains pairs of ($label$, $number$) associated to all correct processes.
- AP^*-$accuracy$: If a process crashes, the $label$ of this process and the corresponding $number$ to this $label$ will be permanently deleted from a_p^*.

More formally:

$S(label)^\tau = \{p_i : (label, -) \in a\_p_i^{*\tau}\}$. $S(label)^\tau$ is the set of all processes who have known $label$ at time τ.

- AP^*-$completeness$: $\exists\tau \in \mathbb{N}$, $\forall p_i \in Correct$, $\forall\tau' \geq \tau$, $\forall(label, number) \in a\_p_i^{*\tau'}$: $|S(label)^{\tau'} \cap Correct| = number$.
- AP^*-$accuracy$: $\forall p_i$, $p_j \in \Pi$, $p_i \in Correct$, $p_j \in Faulty$, $\exists\tau$, $\forall\tau' \geq \tau$: $(label_j, number_j) \notin a\_p_i^{*\tau'}$.

Note that eventually the number of pairs ($label$, $number$) output is equal to the number of correct processes. Moreover, the assignment of labels does not break the anonymity of the system, because labels are assigned and counted in the failure detector implementation, and no process knows the mapping relationship between labels and processes neither in the Reliable Broadcast layer nor in the failure detector layer.

B. Quiescent Reliable Broadcast Algorithm in $AAS\_F_{n,t}[AP^*]$

With failure detector AP^*, the first sub-problem (determining the set of all correct processes in the system) has been solved. The second sub-problem (confirming that a message has been received by all correct processes) can be solved by making every process broadcast an "ACK" message when it receives a "MSG" message. Based on this, a quiescent Reliable Broadcast algorithm in $AAS\_F_{n,t}[AP^*]$ is given in Figure 2.

Description of the algorithm:

The algorithm works as follows. Now every process p_i manages four sets, initially empty: MSG_i, $RB\_DELIVERED_i$, MY_ACK_i (which records all tag_ack generated by p_i), and ALL_ACK_i (which records all tag_ack received by p_i). Similarly to the algorithm of Figure 1, when p_i calls $RB\_broadcast_i(m)$ (line 4), its $random_i()$ generates a random tag for m firstly (line 5). Then, p_i inserts (m, tag) into MSG_i (line 6), so that m will be broadcast periodically to all processes in Task 1 (lines 49-51).

When $receive_i(MSG, m, tag)$ is executed (line 7), p_i inserts (m, tag) into MSG_i if this is the first reception of m (lines 7-12). After that, p_i inserts (m, tag) into $RB\_DELIVERED_i$ and generates a random tag_ack. Then, p_i broadcasts a reception acknowledgment message of (m, tag), which is composed of both tag_ack and $label$ information (read from $a\_p_i^*$). After that, p_i delivers m (lines 16-23). Otherwise, i.e., if an acknowledgment message of (m, tag) is recorded in MY_ACK_i (line 13), then it means that m has already been delivered by p_i. In that case, p_i just broadcasts the recorded acknowledgment message of (m, tag), but with the updated $label$ information from $a\_p_i^*$ (lines 14-15).

When process p_i receives an acknowledgment message (ACK, m, tag, tag_ack, $labels_j$) from process p_j (note that p_j could be p_i itself), there are three cases to consider:

[1]We use the complement of a suspicion to describe strong completeness.

```
1  Initialization
2    sets MSG_i, RB_DELIVERED_i, MY_ACK_i, ALL_ACK_i empty
3    activate Task 1
4  When RB_broadcast_i(m) is executed
5    tag ← random_i()
6    insert (m, tag) into MSG_i
7  When receive_i(MSG, m, tag) is executed
8    if (m, tag) is not in MSG_i then
9      if (m, tag) is not in RB_DELIVERED_i
10       insert (m, tag) into MSG_i
11     end if
12   end if
13   if (m, tag, tag_ack) is in MY_ACK_i then
14     labels_i ← {label | (label, −) ∈ a_p_i^*}
15     broadcast_i(ACK, m, tag, tag_ack, labels_i)
16   else
17     insert (m, tag) into RB_DELIVERED_i
18     tag_ack ← random_i()
19     insert (m, tag, tag_ack) into MY_ACK_i
20     labels_i ← {label | (label, −) ∈ a_p_i^*}
21     broadcast_i(ACK, m, tag, tag_ack, labels_i)
22     RB_deliver_i(m)
23   end if
24 When receive_i(ACK, m, tag, tag_ack, labels_j) is executed
25   if (m, tag, −, −) is not in ALL_ACK_i then
26     allocate array label_counter_i[(m, tag), −]
27     allocate array all_labels_i[(m, tag), −]
28   end if
29   if (m, tag, tag_ack) is not in ALL_ACK_i then
30     insert (m, tag, tag_ack) into ALL_ACK_i
31     all_labels_i[(m, tag), tag_ack] ← labels_j
32     for each label ∈ labels_j do
33       label_counter_i[(m, tag), label] ← label_counter_i[(m, tag), label] + 1
34     end for
35   else
36     for each label in labels_j but not in all_labels_i[(m, tag), tag_ack] do
37       all_labels_i[(m, tag), tag_ack] ← all_labels_i[(m, tag), tag_ack] ∪ {label}
38       label_counter_i[(m, tag), label] ← label_counter_i[(m, tag), label] + 1
39     end for
40     for each label in all_labels_i[(m, tag), tag_ack] but not in labels_j do
41       all_labels_i[(m, tag), tag_ack] ← all_labels_i[(m, tag), tag_ack] \ label
42       delete label_counter_i[(m, tag), label]
43       for each label in both all_labels_i[(m, tag), tag_ack] and labels_j do
44         label_counter_i[(m, tag), label] ← label_counter_i[(m, tag), label] - 1
45       end for
46     end for
47   end if
48 Task 1:
49   repeat forever
50     for every message (m, tag) in MSG_i do
51       broadcast_i(MSG, m, tag)
52       if each pair of (label, number) ∈ a_p_i^*: label_counter_i[(m, tag), label] = number ∧ all_labels_i[(m, tag), −] = {label | (label, −) ∈ a_p_i^*} then
53         if (m, tag) is in RB_DELIVERED_i then
54           delete (m, tag) from MSG_i
55         end if
56       end if
57     end for
58   end repeat
```

Figure 2. Quiescent Reliable Broadcast in $AAS\_F_{n,t}[AP^*]$ (code of p_i)

- p_i receives for the first time an acknowledgment message of (m, tag) (by checking whether (m, tag) is recorded or not in the set ALL_ACK_i), which also means that this is the first ACK message from process p_j (one tag_ack represents one process). In this case, p_i allocates an array $label\_counter_i[(m, tag), -]$ (used to record the number of processes who have known each $label$ received in this ACK message and related to (m, tag)), and an array $all\_labels_i[(m, tag), -]$ (used to record every $label$ in each ACK message related to (m, tag)) (lines 25-28).
- p_i receives an ACK message coming from a new process (by checking whether (m, tag, tag_ack) is recorded or not in ALL_ACK_i). (Observe that the previous case is naturally included in this case, but this case considers not only the very first ACK but later ACKs from others processes). In this case, p_i first inserts (m, tag, tag_ack) into ALL_ACK_i and $labels_j$ into $all\_labels_i[(m, tag), tag\_ack]$. After that, for each received $label$ in $labels_j$, p_i increases its count number by 1 (1 means that every $label$ is known by the process from which tag_ack has been received) (lines 29-34).
- p_i receives a repeated ACK message (with the same tag_ack) (due to the periodical broadcast of messages to cope with fair lossy channels). There are two mutually

exclusive cases: 1) repeated ACK with "more" (new) label information (lines 36-39); 2) repeated ACK with "less" label information (due to the accuracy property of AP^*, that may need some time to delete a $label$ corresponding to a crashed process) (lines 40-46). In case 1, for each new $label$, p_i inserts it into $all\_labels_i[(m, tag), tag\_ack]$ and increases its count number by 1. In case 2, for each disappeared $label$, p_i deletes it from $all\_labels_i[(m, tag), tag\_ack]$ and its corresponding $label\_counter$. Then, p_i decreases the count number of received $labels$ by 1 (since it was not accurate due to the ACK message from a faulty process).

In Task 1, for each pair of $(label, number)$ in $a\_p_i^*$, if (1) the counted number of each label $label\_counter_i[(m, tag), label]$ is equal to the corresponding output $number$ of $a\_p_i^*$ (which means that p_i has received $number$ different ACKs (tag_ack) of (m, tag)), and (2) the received labels related to (m, tag) $all\_labels_i[(m, tag), -]$ are equal to the output labels of $a\_p_i^*$ $\{label \mid (label, -) \in a\_p_i^*\}$ (which means that the received ACKs are from correct processes) (line 52), together with the fact that (m, tag) has already been $RB\_delivered$, then p_i deletes (m, tag) from the MSG_i set (line 54).

Correctness Proof:

Theorem 2. *The algorithm of Figure 2 implements Reliable Broadcast quiescently in $AAS\_F_{n,t}[AP^*]$.*

Proof: The proofs of the Validity, Agreement and Uniform Integrity properties of Reliable Broadcast are straightforward. We will now prove the quiescence property of the algorithm of Figure 2.

An algorithm is said to be quiescent when eventually no process sends messages. In the algorithm of Figure 2, it is obvious that the broadcast of ACK messages (lines 15 and 21) is caused by the reception of MSG messages (line 7). Hence, we only need to show that the number of broadcasts of MSG messages is finite. Moreover, by nature a faulty process can only broadcast a finite number of times each MSG messages. Hence, the rest of the proof only focuses on showing that each correct process broadcasts a finite number of times each MSG message.

It is easy to see that the broadcast of MSG messages occur only in Task 1. Let us consider two processes p (correct) and q, such that p broadcasts (MSG, m, tag) periodically by Task 1.

- If q is correct, then eventually both p and q receive this MSG due to the fairness property of fair lossy communication channels. p $RB\_delivers$ m when it receives MSG for the first time. Also, by the algorithm q broadcasts $(ACK, m, tag, tag\_ack, label_q)$ every time it receives MSG. By the fairness property of channels, p will receive some of those ACK messages. According to lines 29-47, p will count every $label$ in the received ACK from q, such that $label\_counter_p[(m, tag), label_q]$ = 2 and $label\_counter_p[(m, tag), label_p]$ = 2. From the properties of the failure detector AP^*, the output of AP_p^* is composed of $label$ and $number$ of correct processes, e.g., $[(label_q, 2), (label_p, 2)]$. Then, the condition of line 52 is satisfied, and thus process p deletes (m, tag) from MSG_i and the repeated broadcast of the MSG message is stopped, which proves that this case is quiescent.
- If q has crashed, then p will only receive ACK from itself and together with the accuracy property of AP_p^*, the $label$ and corresponding $number$ of q will eventually and permanently be removed from the output of AP_p^*. Again, the condition of line 52 is satisfied, which proves that this case is quiescent too.

The previous reasoning completes the proof of the quiescence property of the algorithm of Figure 2. ■

V. Conclusion

In this paper, we have studied the implementation of Reliable Broadcast in anonymous asynchronous message passing distributed systems with fair lossy communication channels. We have initially proposed a non-quiescent algorithm, proving that it is possible to implement RB in fair lossy anonymous distributed systems. In this first algorithm, each correct process has to broadcast all $RB\_delivered$ messages forever in order to overcome the message losses caused by the fair lossy communication channels. Then, an anonymous perfect failure detector AP^* has been proposed, which allows stopping eventually the periodical broadcast in order to get a more practical quiescent RB algorithm. Finally, a quiescent RB algorithm is given in the fair lossy anonymous distributed system model enriched with AP^*.

References

[1] F. Schneider, D. Gries, and R. Schlichting. Fault-tolerant broadcast. Science of Computer Programming 4(1), pp. 1–15, 1984.

[2] Vassos Hadzilacos and Sam Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical Report, Cornell University, Ithaca, NY, USA, 1994.

[3] Chang J M, Maxemchuk N F. Reliable broadcast protocols. ACM Transactions on Computer Systems (TOCS), 2(3), pp. 251–273, 1984.

[4] Aguilera M K, Chen W, Toueg S. Heartbeat: A timeout-free failure detector for quiescent reliable communication. Distributed Algorithms, pp. 126–140, Springer Berlin Heidelberg, 1997.

[5] Zohir Bouzid, Pierre Sutra, and Corentin Travers. Anonymous Agreement: The Janus Algorithm. Proc. of the 15th international conference on Principles of Distributed Systems OPODIS'11, pp. 175–190, Springer-Verlag Berlin, Heidelberg, 2011.

[6] S. Arévalo, E. Jiménez, and J. Tang. Fault-tolerant broadcast in anonymous systems. Technical Report, Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, Madrid, Spain, 2014.

[7] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. On the power of anonymous one-way communication. Principles of Distributed Systems, Lecture Notes in Computer Science Volume 3974, pp. 396–411,Springer Berlin Heidelberg, 2006.

[8] R. Bakhshi, W. Fokkink, J. Pang, and J. Van de Pol. Leader Election in Anonymous Rings: Franklin Goes Probabilistic. IFIP International Federation for Information Processing, Volume 273, pp. 57–72, 2008.

[9] P. Fraigniaud, A. Pelc, D. Peleg, and S. Prennes. Assigning labels in an unknown anonymous network with a leader. Distributed Computing, 14(3), pp. 163–183, July 2001.

[10] A. D.Kshemkalyani and M. Singhal. Efficient distributed snapshots in an anonymous asynchronous message-passing system. Journal of Parallel Distributed Computing, 73(5), pp. 621–629, May 2013.

[11] R. Guerraoui and E. Ruppert. What Can Be Implemented Anonymously? Proc. of the 19th International Conference on Distributed Computing (DISC'05), pp. 244–259, Springer, 2005.

[12] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. Distributed Computing, 18(4), pp. 235–253, 2006.

[13] D. Angluin. Local and global properties in networks of processors (extended abstract). Proc. of the 12th Annual ACM Symposium on Theory of Computing (STOC '80), pp. 82–93, ACM New York, 1980.

[14] H. Buhrman, A. Panconesi, R. Silvestri, and P. Vityani. On the importance of having an identity or is consensus really universal?. Distributed Computing, 18(3), pp. 167–175, 2006.

[15] C. Delporte-Gallet, H. Fauconnier and A. Tielmann. Fault-Tolerant consensus in unknown and anonymous networks. Proc. of 29th IEEE International Conference on Distributed Computing Systems (ICDCS'09), pp. 368–375, 2009.

[16] R. Guerraoui and E. Ruppert. Anonymous and fault-tolerant shared-memory computing. Distributed Computing, 20(3), pp. 165–177, 2007.

[17] C. Delporte-Gallet, H. Fauconnier, and H. Tran-the. Homonyms with forgeable identifiers. Proc. of the 19th International Conference on Structural Information and Communication Complexity (SIROCCO'12), pp. 171–182. Springer-Verlag Berlin, Heidelberg, 2012.

[18] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. Proc. of the 10th International Workshop on Distributed Algorithms, pp. 105–122, Springer-Verlag, London, 1996.

[19] Y. Afek, H. Attiya, A. Fekete, M. Fisher, N. Lynch, Y. Mansour, D. Wang, and L. Zuck. Reliable communication over unreliable channels. Journal of the ACM, 41(6), pp. 1267–1297, 1994.

[20] M. Aguilera, S. Toueg, and B. Deianov. Revisiting the weakest failure detector for uniform reliable broadcast. Proc. of the 13th International Symposium on Distributed Computing (DISC'99), pp. 19–33, Bratislava, Slovak Republic, September 1999.

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

[22] Yves-Alexandre de Montjoye, L. Radaelli, V. K. Singh, and A. Pentland. Unique in the shopping mall: On the reidentifiability of credit card metadata, Science, 347(6221), pp. 536–539, 30 January 2015.

[23] Mathis F. H. A generalized birthday problem. SIAM Review, 33(2), pp. 265–270, 1991.

[24] F. Bonnet and M. Raynal. Anonymous asynchronous systems: the case of failure detectors. Proc. of the 24th International Symposium on Distributed Computing (DISC'10), pp. 206–220, Cambridge, MA, USA, September 2010.