Implementing Uniform Reliable Broadcast in Anonymous Distributed Systems with Fair Lossy Channels

Jian Tang*, Mikel Larrea†, Sergio Arévalo‡, and Ernesto Jiménez‡§

*Distributed Systems Laboratory (LSD), Universidad Politécnica de Madrid, 28031 Madrid, Spain
Email: tjapply@gmail.com
†University of the Basque Country UPV/EHU, 20018 San Sebastián, Spain
Email: mikel.larrea@ehu.es
‡Universidad Politécnica de Madrid, 28031 Madrid, Spain
§Prometeo Researcher, EPN, Ecuador
Email: {sergio.arevalo, ernes}@eui.upm.es

***Abstract*—Uniform Reliable Broadcast (URB) is an important abstraction in distributed systems, offering delivery guarantee when spreading messages among processes. Informally, URB guarantees that if a process (correct or not) delivers a message m, then all correct processes deliver m. This abstraction has been extensively investigated in distributed systems where all processes have different identifiers. Furthermore, the majority of papers in the literature usually assume that the communication channels of the system are reliable, which is not always the case in real systems.**

In this paper, the URB abstraction is investigated in anonymous asynchronous message passing systems with fair lossy communication channels. Firstly, a simple algorithm is given to solve URB in such system model assuming a majority of correct processes. Then a new failure detector class $A\Theta$ is proposed. With $A\Theta$, URB can be implemented with any number of correct processes. Due to the message loss caused by fair lossy communication channels, every correct process in this first algorithm has to broadcast all URB_delivered messages forever, which makes the algorithm to be non-quiescent. In order to get a quiescent URB algorithm in anonymous asynchronous systems, a perfect anonymous failure detector AP^* is proposed. Finally, a quiescent URB algorithm using $A\Theta$ and AP^* is given.

***Index Terms*—Fault-tolerance, Uniform Reliable Broadcast, Message Passing System, Anonymous System, Asynchronous System, Fair Lossy Channel, Failure Detector, Quiescence.**

I. Introduction

The *broadcast communication* abstraction plays an important role in fault-tolerant distributed systems. It is used to disseminate messages among a set of processes, and it has several different forms according to its quality of service [1]. *Best effort broadcast* with two communication operations, namely $send()$ and $receive()$, guarantees that all correct processes will deliver a message if and only if the sender is correct. That is to say, this abstraction does not offer any delivery guarantee if the sender crashes, which may lead to an inconsistent view of the system state by different processes. To avoid this non-determinism in the delivery when the sender may crash, *Reliable Broadcast* (RB) with *RB-broadcast()* and *RB-deliver()* operations was introduced [2], offering some degree of delivery guarantee. In short, RB is a broadcast service that requires that all correct processes deliver the same set of messages, and that all messages sent by correct processes must be delivered by all correct processes. Note that RB only requires the correct processes to deliver the same set of messages, which still may cause inconsistency problems when a process *RB-delivers* a message and then crashes. In order to avoid those inconsistencies, the strongest abstraction *Uniform Reliable Broadcast* (URB) was proposed by Hadzilacos and Toueg ([3], [4], [5]). *Uniform Reliable Broadcast*, with *URB-broadcast()* and *URB-deliver()* operations, guarantees that if a process (no matter correct or not) delivers a message m, then all correct processes deliver m.

The services mentioned above have been extensively studied in the non-anonymous system model, where each process has a unique identifier, usually assuming that communication channels are reliable (if a process p sends a message to a process q, and q is correct, then q eventually receives m) or quasi-reliable (if a process p sends a message to a process q, and the two processes are correct, then q eventually receives m) [6]. However, real channels are neither always reliable nor quasi-reliable, most of them are unreliable (e.g., fair lossy, which means that if a message is sent an arbitrary but finite number of times, there is no guarantee on its reception, it can lose an infinite number of messages [7]). In this regard, several works have addressed the construction of reliable channels

This research is partially supported by the Community of Madrid, under grant CLOUD4BIGDATA (S2013/ICE-2894), the Spanish Research Council, under grants TIN2013-41123-P and TIN2013-46883-P, the Basque Government, under grant IT395-10, the University of the Basque Country UPV/EHU, under grant UFI11/45, the scholarship of Chinese Scholarship Council, and SENESCYT, Ecuador.

over unreliable channels in non-anonymous systems ([7], [8]).

As far as we know, the first research on anonymous systems was conducted by Angluin [9], which led to the works of Yamashita and Kameda ([10], [11]). Then, several papers appeared in this field, e.g., ring anonymous networks and shared memory anonymous systems ([12], [13], [14], [15]). In [16], the reliable broadcast abstraction has been studied in anonymous systems assuming reliable channels.

In classic message passing distributed systems, processes communicate with each other by sending and receiving 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 [17]. However, all these rules have to be changed in anonymous systems. In this paper, each process has a $broadcast()$ communication primitive, with which a process can send a message to all processes (including itself).

Our Contributions This work is devoted to the study of the Uniform Reliable Broadcast (URB) abstraction in anonymous asynchronous message passing distributed systems where processes may crash and communication channels are fair lossy. There are four main contributions in this paper:

- A simple, non-quiescent uniform reliable broadcast algorithm in such a system model assuming a majority of correct processes, which proves that URB can be solved in anonymous asynchronous message passing distributed systems.
- An impossibility result on solving URB without a majority of correct processes.
- Two new classes of anonymous failure detectors $A\Theta$ and AP^*.
- A quiescent uniform reliable broadcast algorithm using $A\Theta$ and AP^*, which does not require a majority of correct processes.

Roadmap This paper is organized as follows. The system model and several definitions are presented in Section 2. A simple and non-quiescent algorithm implementing uniform reliable broadcast is proposed in Section 3 under the condition of a majority of correct processes. Then, in Section 4, an impossibility result on solving uniform reliable broadcast without the condition of a majority of correct processes is given. In order to circumvent this impossibility result and make the algorithm quiescent, two classes of failure detectors $A\Theta$ and AP^* are proposed in Section 5. Then, a quiescent uniform reliable broadcast algorithm with $A\Theta$ and AP^* is given in the Section 6. Finally, the conclusions are presented in Section 7.

II. System Model and Definitions

In this paper, the anonymous asynchronous distributed system is considered as a system in which processes have no identifiers and communicate with each other via a completely connected network with fair lossy communication channels. Two primitives are used in this system to send and receive messages: $broadcast(m)$ and $receive(m)$. We say that a process p_i broadcasts a message m when it invokes $broadcast_i(m)$. Similarly, a process p_i receives a message m when it invokes $receive_i(m)$.

Process The anonymous asynchronous distributed system is formed by a set of n anonymous processes, denoted as $\Pi = \{p_i\}_{i=1,\ldots,n}$, such that its size is $|\Pi| = n$, i $(1 \leq i \leq n)$ is the index of each process of the system. All processes are anonymous, that means they have no identifiers and execute the same algorithm. The index i of process cannot be known by any process of the system. We just use it as a notation like $p_1, \cdots, p_n$ to simplify the description of the algorithms. Furthermore, all processes are asynchronous, that is, there is no assumption on their respective speeds.

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

Failure model 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. We also assume that at least one correct process exists in the system (i.e., $t \leq n{-}1$).

Communication Each pair of processes are connected by bidirectional fair lossy communication channels. 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. In anonymous system, when a process receives a message, it cannot determine who is the sender of this message.

Fair Lossy Channel A channel between two processes p and q is called as fair lossy channel if it satisfies the following properties [18]:

- 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.

Uniform Reliable Broadcast Uniform Reliable Broadcast offers complete delivery guarantees when spreading messages among processes. That is, when a process delivers a message m, then all correct processes have to deliver it. It is also defined in terms of two primitives: $URB\_broadcast(m)$ and $URB\_deliver(m)$. They satisfy the following three properties:

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

Failure Detector A failure detector is a module that provides each process a read-only local variable containing failure information (may be unreliable) of processes. It can be divided into different classes according to the quality of the provided failure information. It was introduced in [19].

Notation The system model is denoted by $AAS\_F_{n,t}[\emptyset]$ or $AAS\_F_{n,t}[D]$. AAS_F is an acronym for anonymous asynchronous message passing distributed systems with fair lossy communication channels; $\emptyset$ means there is no additional assumption, D means the system is enriched with a failure detector class of 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 Uniform Reliable Broadcast in $AAS\_F_{n,t}[t < n/2]$

In this section, a simple implementation algorithm of uniform reliable broadcast under the condition of a majority of correct processes is proposed. The system model of this section is denoted by $AAS\_F_{n,t}[t < n/2]$.

As far as we know, the implementation of the URB abstraction in the classic (non-anonymous) asynchronous systems with a majority of correct processes is simple. In order to ensure the URB termination property, the construction relies on one condition: a message m can be locally URB-delivered to the upper application layer when this m has been received by at least one non-faulty process. As $n > 2t$, this means that, without risking to be blocked forever, a process may URB-deliver m as soon as it knows that at least $t+1$ processes have received a copy of m. Obviously, this condition is also needed to be satisfied in the anonymous distributed systems. However, there is no easy way to identify who is the correct process that has received m in the anonymous asynchronous message passing distributed systems due to the fact that all processes have no identifiers. In order to solve this difficulty, the idea of implementing URB in $AAS\_F_{n,t}[t < n/2]$ is as follows: 1) to add a unique label (tag) to each message by its sender before it to be broadcast; 2) to add a unique label (tag_ack) to each acknowledgment message (denoted by ACK) when a process receives a message.

With the idea mentioned above, the $URB\_deliver$ condition can be expressed in an equivalent way: each process can deliver a message m if it has received a majority of distinct $ACKs$ of m. Together with the condition of a majority of correct processes, it is guaranteed that at least one correct process has received m.

Description of the algorithm: Algorithm 1 is the implementation algorithm of uniform reliable broadcast abstraction in $AAS\_F_{n,t}[t < n/2]$. In this algorithm, two types of messages are transmitted: MSG (a message needs to be $URB\_delivered$) and ACK (reception acknowledgment of a message). Each process manages a random function $random()$ and four local sets:

- MSG_i, initialized to empty, records all messages that it has received.
- $URB\_DELIVERED_i$, initialized to empty, records all $URB\_delivered$ messages.
- MY_ACK_i, initialized to empty, records all acknowledgment messages of each message generated by itself.
- ALL_ACK_i, initialized to empty, records all acknowledgment messages of each message it has received (from any process).

Algorithm 1 Uniform Reliable Broadcast in $AAS\_F_{n,t}[t < n/2]$ (code of p_i)

```
 1 Initialization
 2   sets $MSG_i$, $MY\_ACK_i$, $ALL\_ACK_i$, $URB\_DELIVERED_i$ empty
 3 activate Task 1

 4 When $URB\_broadcast_i(m)$ is executed
 5   $tag \leftarrow random_i()$
 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 $(m, tag, tag\_ack)$ is in $MY\_ACK_i$ then
12     $broadcast_i(ACK, m, tag, tag\_ack)$
13   else
14     $tag\_ack \leftarrow random_i()$
15     insert $(m, tag, tag\_ack)$ into $MY\_ACK_i$
16     $broadcast_i(ACK, m, tag, tag\_ack)$
17   end if

18 When $receive_i(ACK, m, tag, tag\_ack)$ is executed
19   if $(m, tag, tag\_ack)$ is not in $ALL\_ACK_i$ then
20     insert $(m, tag, tag\_ack)$ into $ALL\_ACK_i$
21   end if
22   if there is a majority of $(m, tag, -)$ in $ALL\_ACK_i$
       then
23     if $(m, tag)$ is not in $URB\_DELIVERED_i$ then
24       insert $(m, tag)$ into $URB\_DELIVERED_i$
25       $URB\_deliver_i(m)$
26     end if
27   end if

   Task 1:
28   repeat forever
29     for every message $(m, tag)$ in $MSG_i$ do
30       $broadcast_i(MSG, m, tag)$
31     end for
32   end repeat
```

Then, let us consider a process p_i to simplify the description. At the beginning, p_i initializes all its sets into empty and activates Task 1 (lines 1-3).

When p_i calls $URB\_broadcast(m)$, it assigns a unique random tag to this message m, and inserts this pair of (m, tag) into MSG_i (lines 4-6). Then, this (m, tag) is broadcast forever in the Task 1 to propagate it to all processes (lines 28-32).

When p_i receives a message (MSG, m, tag) (may come from itself or others process), there are three cases:

- If p_i receives (MSG, m, tag) from itself for the first time (i.e., if this (m, tag) has already existed in MSG_i, but its ACK message (m, tag, tag_ack) does not exist in MY_ACK_i). This process will go to execute line 14, generates a random tag_ack to tag the acknowledgment message of (m, tag). Then, p_i inserts this acknowledgment message (m, tag, tag_ack) into its sets MY_ACK_i, and broadcasts (ACK, m, tag, tag_ack) to all processes to acknowledge the reception of (m, tag) (lines 15,16). This tag_ack is unique for each pair of (m, tag), which means tag_ack cannot be changed for the same pair of (m, tag) once it is generated. The local set MY_ACK_i is used to maintain this uniqueness, to distinguish the tag_ack generated by itself from the received tag_ack from others process.
- If p_i receives (MSG, m, tag) from others process for the first time (i.e., if this (m, tag) does not in MSG_i, neither its ACK message (m, tag, tag_ack) does not exist in MY_ACK_i). It inserts this message into MSG_i (lines 8, 9). Then, like the first case, p_i generates a random tag_ack to tag the acknowledgment message of (m, tag) (line 14). Then, p_i inserts this acknowledgment message (m, tag, tag_ack) into MY_ACK_i, and broadcasts (ACK, m, tag, tag_ack) to all processes to confirm the reception of (m, tag) (lines 15,16).
- If p_i has received a (m, tag) already (i.e., if this (m, tag) has already existed in MSG_i and its ACK message (m, tag, tag_ack) also exists in MY_ACK_i), it re-broadcasts the identical acknowledgment message (ACK, m, tag, tag_ack) to all processes in order to confirm the reception of (m, tag) to overcome the message lost caused by the fair lossy communication channels (lines 11,12).

When p_i receives an acknowledgment message (denoted by ACK) for the first time, it inserts this ACK message to its set ALL_ACK_i (lines 19-21).

When p_i receives a majority of acknowledgment messages (m, tag, tag_ack) of (m, tag) (more than $n/2$ different tag_ack), and this m with tag has not been $URB\_delivered$ yet, then p_i $URB\_deliver$ m for one time (lines 22-25).

Theorem 1 *The algorithm 1 guarantees the property of URB.*

The correct proof of this theorem is straightforward, and it can be found in [21].

Remark: The algorithm 1 can fulfill a fast $URB\_deliver()$ of a message due to the property of fair lossy communication channels and the asynchrony of the system. For example, a process may receive a majority of acknowledgment messages (ACK, m, tag, tag_ack) and $URB\_deliver$ m (according to line 22). Hence, this $URB\_deliver$ is earlier than the reception of (MSG, m, tag). However, this does not violate the property of URB, even if this fast deliver process crashes after $URB\_deliver$ m. Because this fast deliver process has received a majority of acknowledgment messages of m before $URB\_deliver()$ m, which means a majority of processes have received this m (different processes generate distinct ACKs to the same m). Because together with the condition that there is a majority of correct processes, it is guaranteed that at least one correct process has received m. Then, this correct process will broadcast m forever guaranteeing that all correct processes will receive m. If the fast deliver process is correct, it will receive m eventually from others correct process.

It is necessary to generate a unique tag to each MSG and a unique tag_ack to each ACK in this algorithm. However, it is possible that one random value can be shared by two messages only if one is MSG type and another one is ACK type.

IV. An Impossibility Result

In this section, it is proved that the assumption of a majority of correct processes in the algorithm 1 is a necessary condition to solve URB in $AAS\_F_{n,t}[\emptyset]$ if without any other additional assumption.

Theorem 2 It is impossible to solve URB in $AAS\_F_{n,t}[\emptyset]$ without a majority of correct processes.

Proof: The proof is by contradiction, let us suppose there exists an algorithm A that solves URB in $AAS\_F_{n,t}[t \geq n/2]$. Then we divide all processes in the system into two subsets S_1 and S_2, such that $| S_1 |=\lceil n/2 \rceil$ and $| S_2 |=\lfloor n/2 \rfloor$. Now, we consider two runs: R_1 and R_2.

- Run R_1. In this run, all processes of S_2 crash initially, and all the processes in S_1 are non-faulty. Moreover, if a process in S_1 issues $URB\_broadcast(m)$. Due to the very existence of the algorithm A, every process of S_1 $URB\_delivers$ m.
- Run R_2. In this run, all processes of S_2 are non-faulty, and no process of S_2 ever issues $URB\_broadcast()$. The processes of S_1 behave as in R_1: a process issues $URB\_broadcast(m)$, and they all $URB\_deliver$ m. Moreover, after it has $URB\_delivered$ m, every process of S_1 crashes, and all messages ever sent by the process of S_1 are lost, neither has been received by a process of S_2. Hence, no process in S_2 will $URB\_deliver$ m.

It is easy to see that all processes of S_1 cannot distinguish run R_2 from run R_1 before they $URB\_deliver$ m, as they did in run R_1. Then after that all processes in S_1 are crashed, together with the fair lossy channel, no process in S_2 has received m. This violates the uniform agreement of URB, so the algorithm A does not exist. We complete the proof of Theorem 2. ■

V. Two Failure Detector Classes

In this section, two classes of failure detector are proposed. One is used to circumvent the impossibility mentioned above, one is used to make the algorithm 1 to be quiescent.

A. Failure Detector $A\Theta$

Following the previous impossibility result, one question appears naturally, that is, what extra information is needed if the uniform reliable broadcast abstraction is implemented under the assumption that any number of processes can crash? The answer is that the confirmation of a message m has been received by at least one correct process p_j before a process $p_i (i \neq j)$ $URB\_deliver$ this m. Thanks to the failure detector that was proposed by S. Toueg, this confirmation can be guaranteed by the usage of the (unreliable) failure information provided by it. In this section, we try to circumvent such an impossibility result by using the failure detector.

In non-anonymous systems, failure detector Θ is considered as the weakest one to solve URB. It is defined as that it always trust at least one correct process (accuracy) and eventually every correct process do not trust any crashed process (completeness) [18]. The counterpart of this Θ in anonymous distributed system is named as $A\Theta$. Then, we try to define $A\Theta$ in the anonymous asynchronous distributed systems.

$A\Theta$ provides the same failure information as Θ if each process has a unique identifier. However, it is impossible to give such information in anonymous systems because each process has no identifier. So, the key point to define $A\Theta$ is how to identify every process without breaking the anonymity of the system. We are inspired by the definition of failure detector class of $A\Sigma$, which was introduced by F. Bonnet and M. Raynal [20], to define the $A\Theta$. This $A\Theta$ provides each process with a read-only local variable a_theta_i that contains several pairs of ($label$, $number$), in which one $label$ represents a temporary identifier of one process and $number$ represents the number of correct processes who have known this $label$. For example, process p_j's local variable a_theta_j = {($label_1$, $number_1$), ... , ($label_i$, $number_i$), ... , $(label_n, number_n)$} if there are n processes in the system. A $label$ is assigned randomly to each process without breaking the anonymity of the system due to the fact that each process does not know the mapping relationship between a $label$ and a process (even itself).

The definition of $A\Theta$ is given as follows:

- $A\Theta$-$completeness$: There is a time after which the output variable a_theta permanently contains pairs of ($label$, $number$) associated to all correct processes.
- $A\Theta$-$accuracy$: If there is a correct process, then at every time, all pairs of $(label, number)$ outputted by failure detector $A\Theta$ hold that for all subset T of size $number$ of processes that know a $label$ contains at least one correct process (i.e., for each $label$, there always exists one correct process in the output set of $number$ processes that knows this label).

Then we give this definition more formally.
Formal definition of $A\Theta$:

$S(label)$={i | $\exists\ \tau \in \mathbb{N}$: $(label, -) \in a\_theta_i^{\tau}$}. $S(label)$ is a set of all processes who have known the $label$.

• $A\Theta$-$completeness$: $\exists\ \tau \in \mathbb{N}$, $\forall\ i \in Correct$, $\forall\ \tau' \geq \tau$, $\forall$ $(label, number) \in a\_theta_i^{\tau'}$: $|S(label) \cap Correct| = number$.

• $A\Theta$-$accuracy$: $Correct \neq \emptyset \Longrightarrow \forall\ \tau \in \mathbb{N}$, $\forall\ i \in \Pi$, $\forall$ $(label, number) \in a\_theta_i^{\tau}$: $\forall\ T \subseteq S(label)$, $|T|$ = $number$: $T \cap Correct \neq \emptyset$.

B. Failure Detector AP^*

An algorithm is quiescent means that eventually no process sends or receives messages. Hence, it is obvious that the algorithm 1 is a non-quiescent algorithm since every correct process has to broadcast all $URB\_delivered$ messages forever. However, a quiescent algorithm is more valuable and practical in the real systems. In this section, we try to solve this quiescent problem.

The intuitive idea to obtain a quiescent URB algorithm is to terminate the forever broadcast in the algorithm 1. According to the property of uniform reliable broadcast, this forever broadcast can be stopped when a message has been $URB\_delivered$ by all correct processes (i.e., delete messages that have been $URB\_delivered$ by all correct processes from the set MSG). In order to realize this idea, another failure detector AP^* is needed to enrich the system model to provide the information of who are correct processes in the system.

Anonymous perfect failure detector AP^* provides each process with a read-only local variable a_p^* that contains several pairs of ($label$, $number$), which is similar to the failure detector $A\Theta$.

To be more clearly, the definition of AP^* is given as follows:

- AP^*-$completeness$: There is a time after which the output variable 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 the $label$ will be eventually and permanently deleted from the output variable a_p^*.

Eventually the number of pairs of ($label$, $number$) is equal to the number of correct processes.

Let us define AP^* more formally:

$S(label)$={i | $\exists\ \tau \in \mathbb{N}$: $(label, -) \in a\_p^{*\tau}_{i}$}, which is the set of all processes who have known this $label$ at time τ according to $a\_p_i^*$.

• AP^*-$completeness$: $\exists\ \tau \in \mathbb{N}, \forall\ i \in Correct, \forall\ \tau' \geq \tau, \forall$ $(label, number) \in a\_p^{*\tau'}_{i}$: $|S(label) \cap Correct| = number$.

• AP^*-$accuracy$: $\forall$ i, j $\in \Pi$, i $\in Correct$, $\exists\ \tau$: j $\in Fault$: $\forall\ \tau' \geq \tau$: ($label_j$, $number_j$) $\notin a\_p^{*\tau'}_{i}$.

VI. Quiescent Uniform Reliable Broadcast in $AAS\_F_{n,t}[A\Theta, AP^*]$

In this section, the anonymous asynchronous distributed system model is enriched by both failure detectors $A\Theta$ and AP^*, denoted by $AAS\_F_{n,t}$[$A\Theta$, AP^*]. The algorithm 2 is the quiescent implementation algorithm of the uniform reliable broadcast abstraction in $AAS\_F_{n,t}$[$A\Theta$, AP^*] under the assumption of any number of processes can crash. We first give a detailed description of it as follows:

Each process initializes its four sets: MSG_i, $URB\_DELIVERED_i$, MY_ACK_i, ALL_ACK_i and activates the Task 1 (lines 1-3). We take a process p_i as an example to simplify the description. When p_i calls $URB\_broadcast(m)$, it generates a random tag to this

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

```
1  Initialization
2    sets $MSG_i$, $URB\_DELIVERED_i$, $MY\_ACK_i$, $ALL\_ACK_i$ empty
3    activate Task 1

4  When $URB\_broadcast_i(m)$ is executed
5    $tag \leftarrow random_i()$
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 $URB\_DELIVERED_i$ then
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     $labels_i \leftarrow \{label \mid (label,\ -) \in a\_theta_i\ \}$
15     $broadcast_i(ACK, m, tag, tag\_ack, labels_i)$
16   else
17     $tag\_ack \leftarrow random_i()$
18     insert $(m, tag, tag\_ack)$ into $MY\_ACK_i$
19     $labels_i \leftarrow \{label \mid (label,\ -) \in a\_theta_i\ \}$
20     $broadcast_i(ACK, m, tag, tag\_ack, labels_i)$
21   end if

22 When $receive_i(ACK, m, tag, tag\_ack, labels_j)$ is executed
23   if $(m, tag, -, -)$ is not in $ALL\_ACK_i$ then
24     allocate array $label\_counter_i[(m,\ tag),\ -]$
25     allocate array $all\_labels_i[(m,\ tag),\ -]$
26   end if
27   if $(m, tag, tag\_ack)$ is not in $ALL\_ACK_i$ then
28     insert $(m, tag, tag\_ack)$ into $ALL\_ACK_i$
29     $all\_labels_i[(m,\ tag),\ tag\_ack] \leftarrow labels_j$
30     for each label $\in labels_j$ do
31       $label\_counter_i[(m,\ tag),\ label] \leftarrow label\_counter_i[(m,\ tag),\ label]$ + 1
32     end for
33   else
34     for each label in $labels_j$ but not in $all\_labels_i[(m,\ tag),\ tag\_ack]$ do
35       $all\_labels_i[(m,\ tag),\ tag\_ack] \leftarrow all\_labels_i[(m,\ tag),\ tag\_ack] \cup \{label\}$
36       $label\_counter_i[(m,\ tag),\ label] \leftarrow label\_counter_i[(m,\ tag),\ label]$ + 1
37     end for
38     for each label in $all\_labels_i[(m, tag), tag\_ack]$ but not in $labels_j$ do
39       $all\_labels_i[(m,\ tag),\ tag\_ack] \leftarrow all\_labels_i[(m,\ tag),\ tag\_ack] \setminus \{label\}$
40       delete $label\_counter_i[(m,\ tag),\ label]$
41       for each $label$ in both $all\_labels_i[(m,\ tag),\ tag\_ack]$ and $labels_j$ do
42         $label\_counter_i[(m,\ tag),\ label] \leftarrow label\_counter_i[(m,\ tag),\ label]$ - 1
43       end for
44     end for
45   end if
46   if $\exists(label,\ number) \in a\_theta_i$: $label\_counter_i[(m,\ tag),\ label] = number$ then
47     if $(m,\ tag)$ is not in $URB\_DELIVERED_i$ then
48       insert $(m,\ tag)$ into $URB\_DELIVERED_i$
49       $URB\_deliver_i(m)$
50     end if
51   end if

Task 1:
52 repeat forever
53   for every message $(m,\ tag)$ in $MSG_i$ do
54     $broadcast_i(MSG, m, tag)$
55     if each pair of $(label,\ number) \in a\_p_i^*$: $label\_counter_i[(m,\ tag),\ label] = number \wedge all\_labels_i[(m,\ tag),\ -] = \{label \mid (label,\ -) \in a\_p_i^*\ \}$ then
56       if $(m, tag)$ is in $URB\_DELIVERED_i$ then
57         delete $(m, tag)$ from $MSG_i$
58       end if
59     end if
60   end for
61 end repeat
```

message m and inserts (m, tag) into set MSG_i (lines 4-6). Then, this m is broadcast to all processes forever in the Task 1 in the form of (MSG, m, tag) (lines 52-54).

When p_i receives a message (MSG, m, tag), it first checks if this (m, tag) has already existed in its MSG_i. If not, then it checks if (m, tag) has already been $URB\_delivered$ (lines 8, 9). If not, p_i inserts this message to MSG_i (line 10). Otherwise, p_i overlook this reception. Then, there are three cases as in the algorithm 1:

- If p_i receives (MSG, m, tag) from itself for the first time (i.e., if this (m, tag) has already existed in MSG_i, but its ACK message (m, tag, tag_ack) does not exist in MY_ACK_i). Then, p_i goes to execute the line 17 that generates a random tag_ack to tag the acknowledgment message of (m, tag). Then, p_i inserts this acknowledgment message (m, tag, tag_ack) into its sets MY_ACK_i, and reads the $label$ information from its failure detector $A\Theta_i$. Then, p_i broadcasts $(ACK, m, tag, tag\_ack, labels_i)$ to all processes to acknowledge the reception of (m, tag) (lines 17-20).
- If p_i receives (MSG, m, tag) from others process for the first time (i.e., if this (m, tag) does not exist in MSG_i or $URB\_DELIVERED_i$, neither its ACK message (m, tag, tag_ack) does not exist in MY_ACK_i). It

inserts this message into MSG_i (line 10). Then, p_i does the same as the case 1 (lines 17-20).

- If p_i has received this (m, tag) already (i.e., if this (m, tag) has already existed in MSG_i or $URB\_DELIVERED_i$ and its ACK message $(m$, tag, $tag\_ack)$ also exists in $MY\_ACK_i)$, it re-broadcasts the identical acknowledgment message but with the updated $label$ information $(ACK, m, tag, tag\_ack, labels_i)$ to all processes in order to confirm the reception of $(m$, $tag)$ to overcome the message lost caused by the fair lossy communication channels (lines 13-15).

When p_i receives an acknowledgment message $(ACK$, m, tag, tag_ack, $labels_j)$ from p_j (could be itself), there are three cases as follows:

- p_i receives the very first ACK message of (m, tag), which also means this is the first time receives an ACK message with tag_ack (by checking $(m$, $tag)$ exists in the set ALL_ACK_i or not), which also means this is the first ACK message from one process (tag_ack represents a process).

  p_i allocates an array $label\_counter_i[(m$, $tag)$, $-]$ (used to record the number of every $label$ that received together with $(m$, $tag)$), and an array $all\_labels_i[(m$, $tag)$, $-]$ (used to records all $label$ in each ACK message of $(m$, $tag)$ (lines 23-25).
- p_i receives an ACK message coming from a new process (by checking $(m$, tag, $tag\_ack)$ exists in ALL_ACK_i or not). (Case 1 is naturally included in case 2, but case 2 considers not only the very first ACK but more later ACKs from others process).

  p_i firstly insert $(m$, tag, $tag\_ack)$ into the set ALL_ACK_i, and $labels_j$ into the array $all\_labels_i[(m$, $tag)$, $tag\_ack]$. After that, for each received $label$ in $labels_j$, p_i increases its count number by 1(lines 30-32) (1 means that every $label$ is known by the process who generates this ACK with tag_ack).
- p_i receives a repeated ACK message (with the same tag_ack) (i.e., one process re-broadcast an ACK due to the fair lossy channel).

  There are two mutually exclusive cases: 1) repeated ACK with more (new) label information (lines 34-37); 2) repeated ACK with less label information (due to the completeness property of $A\Theta$ that it needs some time to delete a $label$ of crashed process) (lines 38-44). In one instance of algorithm 2, only one of these two cases can happen. 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, decreases the count number of repeatedly received $label$ by 1 (miscount this $label$ by 1 due to the ACK from the crashed process).

After counting the $number$ of each $label$, if there exists one pair of $(label$, $number)$ outputted by $A\Theta_i$ satisfies the condition that the counted number of this $label$ $label\_counter_i[(m$, $tag)$, $label]$ is equal to the outputted $number$, then p_i checks this m has been $URB\_delivered$ or not. If not, p_i $URB\_deliver$ m for one time.

In task 1, for each pair of $(label$, $number)$ in the output of AP_i^*, if the condition that the counted number of each $label$ $label\_counter_i[(m$, $tag)$, $label]$ is equal to the corresponding $number$ (means it has received $number$ different ACKs(tag_ack) of $(m$, $tag)$) and the set of received label related to $(m$, $tag)$ $all\_labels_i[(m$, $tag)$, $-]$ is equal to the outputted label set of AP_i^* $\{label \mid (label, -) \in a\_p_i^*\}$ (means the received ACKs (tag_ack) are from the correct processes) is satisfied (line 55), and together with the fact that $(m$, $tag)$ has already been $URB\_delivered$, then, p_i deletes $(m$, $tag)$ from its MSG_i set (line 57).

Lemma 1: *If a correct process broadcasts a message m, then it eventually deliver m.* (*Validity*)

Proof: Let us consider a non-fault process p_i broadcasts m. A unique random tag is assigned to this message m (line 5), then inserts $(m$, $tag)$ into the set MSG_i to be $broadcast$ a bounded but unknown times (until the condition of line 55 is satisfied) in Task 1 (lines 52-54). Together with the fairness property of fair lossy channel, all correct processes (include p_i) will receive this m eventually.

Then, when a correct process receives $(MSG$, m, $tag)$ for the first time, it generates a second unique tag_ack to the corresponding acknowledgment message and broadcasts it to all processes. Due to the bounded but unknown times of $broadcast(MSG$, m, $tag)$ in the Task 1 of p_i, each correct process receives it for a bounded but unknown times. Hence, each process $broadcast$ an acknowledgment message for a bounded but unknown times too. The same reason of the fairness property of the communication channels, p_i will receive all acknowledgment messages of (m, tag) from correct processes. Then, it is obvious that the condition of line 46 is satisfied, and p_i delivers m. We complete the proof of Lemma 1. ■

Lemma 2: *If some process deliver a message m, then all correct processes eventually deliver m.* (*Uniform Agreement*)

Proof: To prove this Lemma, we consider the following two cases:

Case 1: A message *m* is delivered by a correct process.

Suppose this correct process is p_i, then according to lines 52-54 of Task 1, p_i will broadcast m for a bounded but unknown times (until the condition of line 55 is satisfied) to all processes. With the fairness property of the channels, all correct processes will eventually receive m. Then, all correct process will do the same as p_i to broadcast this m a bounded but unknown times. Together with the Lemma 1, all correct processes eventually deliver this m.

Case 2: A message *m* is delivered by a crashed process.

The condition of line 46 was satisfied before this crashed process deliver *m*. Due to the accuracy property of $A\Theta$, at least one correct process has received this m. Then, this correct process will broadcast m for a bounded but unknown times (until the condition of line 55 is satisfied). Together with

Lemma 1, it is obvious that all correct processes will deliver m.

Following case 1 and 2, we can see that Lemma 2 is correct. ■

Lemma 3: *For every message m, every process delivers m at most once, and only if m was previously broadcast by sender(m).* (*Uniform Integrity*)

Proof: It is easy to see that any message m was previously broadcast by its sender, because each process only forwards messages it has received and the fair lossy channel does not create, duplicate, or garble messages.

To prove a message only be delivered for at most one time, let us observe that two kinds of tags exist in the system: one is used to label the message itself; one is used to label the acknowledgment of this message. The set MY_ACK_i is used to guarantee that each process broadcasts the identical acknowledgment message to the same (m, tag) (line 18). The set $URB\_DELIVERED_i$ to record all messages that have delivered (line 48).

Even each message is broadcast for a bounded but unknown times (until the condition of line 55 is satisfied) and will be received by every correct process for a bounded but unknown times (lines 52-54), one message can not be modified or relabeled as a new message due to these tags and sets mentioned above. Moreover, every message is checked whether it has already existed in its set $URB\_DELIVERED_i$ (line 47) before $URB\_deliver$ it. With those mechanisms, it is certain that no message m will be delivered more than once. Hence, the proof is completed. ■

Theorem 3 *Algorithm 2 is a quiescent implementation of the uniform reliable broadcast communication abstraction in* $AAS\_F_{n,t}[A\Theta, AP^*]$.

Proof: From Lemma 1, 2 and 3, it is easy to see that algorithm 2 is the implementation of the uniform reliable broadcast. Then, it is only necessary to prove the algorithm 2 satisfies the quiescent property.

An algorithm is quiescent means that eventually no process sends or receives messages. In algorithm 2, it is obvious that the broadcast of ACK message is invoked by the reception of MSG message. Hence, the proof is reduced to show that the broadcast times of MSG message is finite. Moreover, a faulty process only broadcast a finite times of MSG message. Hence, the rest of this proof focus on that each correct process broadcast a finite times of MSG message.

It is easy to see that the broadcast of MSG only exist in Task 1. Let us consider two processes p (correct) and q that p broadcast (MSG, m, tag) to q a bounded but unknown times (p repeat broadcast (MSG, m, tag) until the condition of line 55 is satisfied).

- If q is correct, then eventually both p and q receive this MSG for a bounded but unknown times due to the fairness property of fair lossy communication channels, then p delivers m only once when the first reception of m. Since q broadcast $(ACK, m, tag, label_q)$ to p when each time it receives MSG, q broadcast ACK to p for the same times as the reception times of MSG. (p also can receive MSG from itself and broadcast its ACK. Here, we only take q as an example.) By the fairness property of channels, p receives a bounded but unknown times of ACK. According to lines 22-51, p has to count every $label$ existed in the received ACK from q and itself, as $label\_counter_p[(m, tag)$, $label_q]$=2, $label\_counter_p[(m, tag)$, $label_p]$=2. Together by the property of the failure detector AP^*, the output of AP^*_p is composed by $label$ and $number$ of correct processes that is $[(label_q, 2)$, $(label_p, 2)]$. Then, the condition of line 55 is satisfied that (m, tag) is deleted from MSG and p stops the repeated broadcast of (MSG, m, tag), which proves this case is quiescent.
- If q is faulty. Then, p only receives ACK from itself and together with the accuracy property of AP^*_p, the $label$ and corresponding $number$ of q will eventually and permanently removed from the output of AP^*_p. Hence, it is trivial that the condition of line 55 is satisfied, which proves this case is quiescent too.

Hence, according to the description mentioned above, we complete the proof. ■

VII. Conclusions

In this paper, we have studied how to implement the uniform reliable broadcast abstraction in anonymous asynchronous message passing distributed systems with fair lossy communication channels. A non-quiescent algorithm with the assumption of a majority of correct processes is proposed firstly. Then, in order to obtain a quiescent algorithm and to circumvent the impossibility result of implementing URB without the assumption of a majority of correct processes, two classes of failure detectors are given and used. Finally, the quiescent uniform reliable broadcast algorithm is proposed.

References

[1] C. Cachin, R. Guerraoui, and L. Rodrigues. Reliable and secure distributed programming. Springer (second edition), 2011.

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

[3] V. Hadzilacos. Issues of fault tolerance in concurrent computation. Ph.D thesis, Harvard University, 1984.

[4] V. Hadzilacos, and S. Toueg. Fault tolerant broadcast and related problems. S.J. Mullender (Ed.), Distributed Systems. New York: ACM Press & Addison-Wesley, 1993.

[5] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report 94-1425, 83 pages, Cornell University, Ithaca (USA), 1994.

[6] A. Schiper. Failure detection vs group membership in fault-tolerant distributed systems: hidden trade-offs. Proceedings of the Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, pp. 1–15, Springer-Verlag, London, 2002.

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

[8] Y. Afek, H. Attiya, A. D. 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.

[9] D. Angluin. Local and global properties in networks of processors (extended abstract). Proceedings of the twelfth annual ACM symposium on Theory of computing (STOC '80), pp. 82–93, ACM New York, 1980.

[10] M. Yamashita, and T. Kameda. Computing on anonymous networks, part I: characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems, 7(1): pp. 69–89, 1996.

[11] M. Yamashita, and T. Kameda. Computing on anonymous networks, part II: decision and membership problems. IEEE Transactions on Parallel and Distributed Systems, 7(1): pp. 90–96, 1996.

[12] 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.

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

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

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

[16] Sergio Arévalo, Ernesto Jiménez, and Jian Tang. Fault-tolerant broadcast in anonymous systems. Technical Report of Deparmento de Sistemas Informáticos, 21 pages, Universidad Politécnica de Madrid, Madrid (Spain), 2014.

[17] D. Angluin, J. Aspnes, D. Eisenstat, 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.

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

[19] Tushar Deepak Chandra and Sam Toueg Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, 43:2, pp. 225–267, March, 1996.

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

[21] Jian Tang, Mikel Larrea, Sergio Arévalo, and Ernesto Jiménez. Implementing Reliable Broadcast in Anonymous Distributed Systems with Fair Lossy Channels. Technical Report of University of the Basque Country UPV/EHU, San Sebastián, Spain, 2014. http://www.sc.ehu.es/acwlaalm/research/EHU-KAT-IK-03-14.pdf.