Brief Announcement: Fault-tolerant Broadcast in Anonymous Distributed Systems with Fair Lossy Communication Channels *

Jian Tang
Distributed Systems Laboratory (LSD)
Technical University of Madrid
28031 Madrid, Spain
tjapply@gmail.com

Mikel Larrea
University of Basque Country UPV/EHU
20018 San Sebastián, Spain
mikel.larrea@ehu.eus

Sergio Arévalo
Technical University of Madrid
28031 Madrid, Spain
sergio.arevalo@eui.upm.es

Ernesto Jiménez
Technical University of Madrid
28031 Madrid, Spain
Prometeo Researcher
Escuela Politécnica Nacional
170515 Quito, Ecuador
ernes@eui.upm.es

ABSTRACT

Fault-tolerant broadcast is a fundamental service in distributed systems, by which processes can communicate with each other consistently and reliably. It has two main forms: Reliable Broadcast (RB) and Uniform Reliable Broadcast (URB). This service has been extensively investigated in non-anonymous distributed systems where processes have unique identifiers, usually assume the communication channels are reliable, which is not always the case in real systems. In this paper, the fault-tolerant broadcast service is studied in an anonymous asynchronous message passing distributed system model with fair lossy communication channels. Firstly, two simple and non-quiescent algorithms implementing RB and URB are given. Secondly, two new classes of failure detectors $A\Theta$ and AP^* are proposed. Finally, with the information provided by $A\Theta$ and AP^*, quiescent algorithms for both RB and URB are given.

Categories and Subject Descriptors

H.3.4 [**System and Software**]: Distributed Systems

General Terms

Algorithms, Design, Theory

Keywords

Anonymous, reliable broadcast, failure detector, quiescent.

*The full version of this work can be found in [6].

PODC'15, July 21–23, 2015, Donostia-San Sebastián, Spain.
ACM 978-1-4503-3617-8 /15/07.
http://dx.doi.org/10.1145/2767386.2767443.

1. INTRODUCTION

Fault-tolerant broadcast is a fundamental service in distributed systems, which is helpful to build dependable applications. This service is used to disseminate messages among a set of processes, and has two forms according to its guarantee of delivery [1]. The weakest form is *Reliable Broadcast* (RB), with *RB-broadcast()* and *RB-deliver()* operations, introduced in [2]. In short, RB is a broadcast service requiring all correct processes to deliver the same set of messages, and this set must include all messages broadcast by correct processes. However, this broadcast form can cause some inconsistencies when a process fails after delivering a message. Hence, a stronger broadcast form called *Uniform Reliable Broadcast* (URB) has been proposed [3], with *URB-broadcast()* and *URB-deliver()* operations, guaranteeing that if a process (no matter correct or incorrect) delivers a message m, then all correct processes must deliver m too.

In this paper, the fault-tolerant broadcast service is studied in the anonymous asynchronous message passing distributed system model with fair lossy communication channels.

2. SYSTEM MODEL AND DEFINITIONS

We consider an anonymous asynchronous message passing distributed system in which processes have no identifiers and communicate with each other via a completely connected network of fair lossy channels. The system is composed of n processes, and denoted by $\Pi = \{p_i\}(i = 1, ..., n)$ (i is the index of each process in the system. This index is just used as a notation to simplify the description of the model and the algorithms, and it is not known by any process).

Processes can fail by crashing. A process that does not crash in a run is *correct* in that run, otherwise it is $faulty$. A process executes its algorithm correctly until it crashes. A crashed process can neither execute any more statements nor recover. There is a global clock whose values are positive natural numbers, which is also used for notation purposes.

Processes communicate with each other by sending and receiving messages through fair lossy communication channels. Formally, a channel between two processes p and q is fair lossy if it satisfies the following properties [4]: If p sends a message m to q an infinite number of times and q is correct, then q eventually receives m from p (Fairness); 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 Integrity). Two basic primitives are used in the 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 $receive_i(m)$. Note that in anonymous distributed systems, when a process receives a message m it cannot determine who is the sender of m.

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 fail.

Reliable Broadcast (RB) is a broadcast form that satisfies the following three properties:

- *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. Uniform Reliable Broadcast (URB) is a broadcast form stronger than RB, satisfying Validity and Uniform Integrity (as RB), and *Uniform* Agreement, defined as follows:

- *Uniform Agreement*: If some process (correct or not) delivers a message m, then all correct processes eventually deliver m.

3. IMPLEMENTATION OF FAULT TOLERANT BROADCAST IN AAS_F

A key point to implement the fault-tolerant broadcast service in AAS_F is how to distinguish a process or a message from others. In this paper, we solve this by making each message to be unique. This idea is easily implemented in non-anonymous systems, usually by using the identifier of the sender process and a sequence number. However, in anonymous systems this method cannot be used due to the fact that processes do not have identifiers. Hence, we propose to use a random function, assigning a different random number to each message as its unique label or *tag*. We assume that each number generated randomly is unique in the system. Note that assigning a label to a message is better than to a process, because there is no way to deduce the messages sent from a given process. Indeed, all processes still maintain their anonymity in the system.

With the idea mentioned above and a simple flooding transmission technique, an implementation algorithm of RB in $AAS\_F[\emptyset]$ and another algorithm implementing URB in $AAS\_F[t<n/2]$ are proposed in our full paper [6]. Both algorithms are non-quiescent, i.e., delivered messages need to be re-broadcast periodically in order to cope with fair lossy communication. Additionally, we show that it is impossible to implement URB without a majority of correct processes. Hence, in the next section we enrich the anonymous system model with two failure detectors in order to implement RB and URB quiescently and with any number of correct processes.

4. TWO CLASSES OF ANONYMOUS FAILURE DETECTORS

The failure detector abstraction was proposed Chandra and Toueg [5], which defined them in terms of completeness and accuracy properties. A failure detector can be seen as a module that gives (possibly unreliable) failure information of processes. Usually, this failure information is composed of the identifiers of processes in non-anonymous distributed systems. However, in anonymous distributed systems processes have no identifier. So, the key point to define and implement a failure detector in an anonymous distributed system is how to identify a process without breaking the anonymity of the system. We were inspired by the definition of the failure detector class $A\Sigma$, which was introduced by Bonnet and Raynal [7], that assigns a random *label* to each process as a temporal identifier. This assignment does not break the anonymity of the system, because this *label* is given in the failure detector layer, and no process knows the mapping relationship between labels and processes.

4.1 Failure Detector $A\Theta$

$A\Theta$ provides each process p_i with a read-only local variable a_theta_i that contains several pairs of $(label, number)$, in which each *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 at time τ $a\_theta_j^\tau = \{(label_1, number_1), ..., (label_i, number_i), ..., (label_n, number_n)\}$ if p_j has known the *label* of n processes of the system. As we said previously, labels are assigned randomly to each process without breaking the anonymity of the system.

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

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

Let us define $A\Theta$ more formally:

$S(label)=\{i \mid \exists\ \tau \in \mathbb{N}: (label, -) \in a\_theta_i^\tau\}$. $S(label)$ is the set of all processes that know *label* at time τ.

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

4.2 Failure Detector AP^*

The anonymous perfect failure detector AP^* provides each process p_i with a read-only local variable $a\_p_i^*$ that contains several pairs of ($label$, $number$), similar to failure detector $A\Theta$. The definition of AP^* is as follows:

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

It is assumed that the $number$ of each $label$ is monotonically non-increasing. Eventually the number of pairs of ($label$, $number$) is equal to the number of correct processes.

Let us define AP^* more formally:

$S(label)$=$\{i \mid \exists\ \tau \in \mathbb{N}$: $(label, -) \in a\_p^{*\tau}{}_i\}$. $S(label)$ is the set of all processes that know $label$ at time τ.

- 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*: $\forall\ i, j \in \Pi, i \in Correct, j \in Faulty, \exists\ \tau, \forall\ \tau' \geq \tau$: $(label_j, number_j) \notin a\_p^{*\tau'}{}_i$.

5. QUIESCENT FAULT TOLERANT BROADCAST IN $AAS\_F[A\Theta,\ AP^*]$

Before solving the quiescence problem, the impossibility result of implementing URB without a majority of correct processes is circumvented firstly with $A\Theta$. To do so, in our algorithm an adequate use of $A\Theta$ ensures that the acknowledge of a message m has been received by at least one correct process before delivering m.

The reason why both the RB and URB algorithms of Section 3 are non-quiescent is that each correct process has to broadcast periodically the messages it has delivered in order to overcome message losses caused by the fair lossy communication channels. Therefore, the intuitive idea to make them to be quiescent is to stop the periodical broadcast when a message has been delivered by all correct processes. The following question arises: how can we confirm that a message has been delivered by all correct processes in an anonymous system? In order to answer this question, we propose to combine two mechanisms: (1) each process broadcasts an acknowledgment message after it has delivered a message m, and waits for the acknowledgment messages of m from all processes (including itself); (2) the failure detector AP^* is used to provide the information of all correct processes. With these two mechanisms, a process p_i can stop the periodical broadcast of message m when it has received the acknowledgment of m from all correct processes. Note that these two mechanisms do not break the anonymity of the system.

Following this approach, a quiescent RB algorithm implemented in $AAS\_F[AP^*]$ and a quiescent URB algorithm in $AAS\_F[A\Theta, AP^*]$ are given in our full paper [6].

6. CONCLUSION

In this paper, the fault-tolerant broadcast service has been studied in anonymous asynchronous message passing distributed systems with fair lossy communication channels. Initially, two non-quiescent algorithms have been proposed: the non-quiescent reliable broadcast (RB) algorithm can be implemented under the assumption of anynumber of correct processes, while the non-quiescent uniform reliable broadcast (URB) algorithm requires either a majority of correct processes or a failure detector of class $A\Theta$. In order to implement the fault-tolerant broadcast service quiescently, a new anonymous perfect failure detector AP^* is proposed. Finally, two quiescent algorithms using AP^* and implementing RB and URB respectively are proposed.

Acknowledgment

We would like to thank the anonymous reviewers. This research is partially supported by the Community of Madrid under grant 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, and the scholarship of Chinese Scholarship Council.

7. 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 and S. Toueg. Fault tolerant broadcasts and related problems. S.J. Mullender (Ed.), Distributed Systems. New York: ACM Press & Addison-Wesley, 1993.

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

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

[6] J. Tang, M. Larrea, S. Arévalo, and E. Jiménez. Fault-tolerant broadcast in anonymous distributed systems with fair lossy channels. Technical Report EHU-KAT-IK-06-14 of the University of the Basque Country UPV/EHU, December 2014, http://www.sc.ehu.es/acwlaalm/research/EHU-KAT-IK-06-14.pdf.

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